ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

TRABAJO FIN DE GRADO

DISEÑO DE UN ROBOT MULTI-ARTICULACIÓN

Grado en Ingeniería Electrónica Industrial y Automática

Autores:

Andrés Jorges Palenzuela

Fabrizio Rodríguez Betancourt

Tutor:

Jonay Toledo Carillo

Julio 2024

RESUMEN

En este Trabajo de Fin de Grado (TFG) se ha desarrollado un prototipo de robot multiarticulado con seis patas, también denominado robot hexápodo o insecto artificial. El robot está equipado con 18 servomotores distribuidos entre sus seis patas tri-articuladas, lo que le proporciona una movilidad avanzada y versátil.

El diseño de la estructura mecánica del robot se realizó utilizando el software Fusion 360, y las piezas se imprimieron en 3D utilizando filamento PLA. El ensamblaje se llevó a cabo con tornillería básica, asegurando una construcción robusta y ligera.

Para el control del robot, se empleó el microcontrolador ESP32, que envía los comandos a través de módulos generadores de señales PWM necesarias para mover los servomotores.

El cálculo de trayectorias y la cinemática inversa se realizaron mediante scripts en Python. La información sobre puntos, ángulos y trayectorias se almacena en un archivo en la memoria del ESP32, que luego es leído por el microcontrolador para ejecutar los movimientos del robot.

Para la alimentación del robot, se utilizaron baterías de Li-ion modelo SAMSUNG INR21700-40T, conectadas en serie para proporcionar la tensión necesaria. Además, se desarrolló una aplicación en App Inventor que se comunica vía Bluetooth con el ESP32 para enviar los comandos de movimiento.

ABSTRACT

In this project, a prototype of a multi-jointed robot with six legs, also known as a hexapod robot or artificial insect, has been developed. The robot is equipped with 18 servomotors, distributed among its six tri-articulated legs, to provide advanced and versatile mobility.

The design was created using Fusion 360 software, and the parts were 3D printed using PLA filament. The assembly was carried out using basic screws, ensuring a robust and lightweight construction.

For the robot's control, the ESP32 microcontroller was used, which sends control commands through modules that generate the PWM signals needed to move the servomotors.

Trajectory calculation and inverse kinematics were carried out using Python scripts. The information about points, angles, and trajectories is stored in a file in the ESP32 memory, which is then read by the microcontroller to execute the robot's movements. For the robot's power supply, SAMSUNG INR21700-40T Li-ion batteries were used, connected in series to provide the necessary voltage. To move the robot, an application was developed in App Inventor, which communicates via Bluetooth with the ESP32 to send movement commands.

AGRADECIMIENTOS

Quiero expresar mi más sincero agradecimiento a todas las personas que han sido fundamentales en la realización de este proyecto.

En primer lugar, a mi familia, especialmente a mis padres y a mi hermano, por su constante atención, cariño y apoyo, no solo en este proyecto, sino a lo largo de toda mi vida. Los valores y enseñanzas que me han inculcado han sido vitales para alcanzar esta meta.

A nuestro tutor Jonay, por aceptar inicialmente nuestra idea de proyecto y por acompañarnos durante todo su desarrollo. Su guía, apoyo y herramientas han sido indispensables para la realización de este trabajo.

A Desiré, por su incondicional apoyo, ánimo y ayuda. Gracias por estar a mi lado en las largas noches de programación y ensamblaje, brindándome siempre tu compañía y motivación.

A los compañeros del equipo de Fórmula Student. En especial a Óscar, Enebie, Aythamy, Himar, Carlos, Pablo y Minerva, por su disposición para ayudar en todo momento y su constante interés en el proyecto.

A mis compañeros y amigos de clase Carolina, Fouad, Jorge, Andrés P. y Sabino por sus ánimos y ayuda.

A mi compañero y pareja de TFG Andrés J., por su dedicación y realizar este trabajo en conjunto. Este proyecto no habría sido posible sin él.

Fabrizio Rodríguez Betancourt

Mirando hacia atrás, me doy cuenta de la importancia del apoyo recibido durante este tiempo. Quisiera dedicarles unas líneas de agradecimiento a todas aquellas personas que han contribuido de alguna manera a la realización de este trabajo.

Quiero expresarles, en primer lugar, mi más profundo agradecimiento por todo lo que han hecho por mí a lo largo de mi vida. A ustedes, papá, mamá y mi querido hermano, les estaré eternamente agradecido no solo por enseñarme el valor del trabajo bien hecho, por cultivar en mí un espíritu crítico y por los valores que me han inculcado, sino también el amor y el apoyo incondicional que siempre me han brindado.

A mi querida familia, gracias por ser ese lugar cálido al que siempre puedo acudir para desconectar y descansar. Su apoyo y cariño incondicional han sido un pilar fundamental en mi vida.

A nuestro tutor Jonay, por apoyar este proyecto y por su disponibilidad. Su guía y apoyo han sido fundamentales para el éxito de este proyecto.

A mi pareja de TFG Fabrizio, por su excelente trabajo y compromiso con el proyecto. Este trabajo no hubiera sido exitoso sin su colaboración.

A mis compañeros de grado Andrés, Sabino, Carolina, Fouad y Jorge. La ayuda que he recibido de ustedes es invaluable y siempre estaré en deuda. El equipo que formamos y el apoyo mutuo que nos brindamos han sido fundamentales para nuestro crecimiento y éxito.

Andrés Jorges Palenzuela

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

Este Trabajo de Fin de Grado (TFG) tiene como objetivo el diseño e implementación de un robot hexápodo. Se consolidarán los conocimientos adquiridos durante el grado en Ingeniería Electrónica Industrial y Automática, principalmente en las asignaturas de robótica, diseño gráfico y electrónica.

El control del robot se realizará mediante el microprocesador ESP32, programado en C++ a través de la plataforma y entorno de desarrollo gratuito Arduino. La elección del ESP32 se debe a su versatilidad y potencia, que permiten manejar múltiples servomotores y realizar los cálculos necesarios a gran velocidad para la navegación y control del robot, junto con los módulos generadores de señales PWM.

Este proyecto también tiene el potencial de servir como material didáctico para futuros alumnos. Además, proporciona una base sólida para que otros estudiantes puedan continuar desarrollándolo, añadiendo nuevas funcionalidades que mejoren las capacidades de movimiento del robot e incorporando algoritmos e instrumentación que permitan la navegación en entornos más complejos y la superación de obstáculos.

1.1 Justificación

La motivación para desarrollar este Trabajo de Fin de Grado radica en el creciente interés por los robots multiarticulados, en particular los hexápodos. Estos robots constituyen un área de estudio fascinante debido a la inspiración que toman de la naturaleza, específicamente de la morfología y el comportamiento de animales e insectos con múltiples patas.

La perfección de la naturaleza se refleja en la evolución de estos seres, que han desarrollado mecanismos de movimiento altamente eficientes y adaptables. La capacidad de estos animales para superar obstáculos y realizar movimientos complejos es una fuente de inspiración para la robótica. Trasladar estas habilidades a un robot hexápodo funcional es un desafío técnico y emocionante.

Analizar y replicar la evolución de estos seres en robots hexápodos proporciona una motivación considerable. Este proyecto pretende investigar cómo la adaptación evolutiva de los animales de múltiples patas puede ser implementada en robots para mejorar su movilidad y funcionalidad.

Asimismo, el proyecto busca ampliar los conocimientos en el campo de la robótica, la programación de microprocesadores, y el desarrollo y montaje de hardware electrónico.

1.2 Metodología

Lo primero que se realiza es un estudio exhaustivo y una búsqueda de información sobre los robots hexápodos. Esto incluye su morfología, los componentes que los constituyen y las características específicas de una pata. Se analiza qué tipo de hardware utilizar, incluyendo motores, controladores y la electrónica necesaria para facilitar la programación e implementación del robot, como los módulos generadores de señales PWM.

Una vez comprendidas las partes que conforman un hexápodo, se estudia la cinemática inversa de las patas por separado, tratándolas como si fueran brazos robóticos manipuladores. Se investiga la forma de caminar y los generadores de trayectorias necesarios para realizar movimientos de paso y rotación.

Posteriormente, se programan e implementan los modelos matemáticos y algoritmos mediante scripts en Python, que generan archivos con la información del movimiento del robot. Estos archivos contienen los datos sobre los puntos de la trayectoria que el robot debe seguir.

A continuación, se implementa el código en C++ en el microcontrolador ESP32, que es capaz de leer esta información y transmitirla a los módulos que controlarán los motores.

Una vez que todo esto está listo, se desarrolla una aplicación móvil utilizando App Inventor, que permite controlar el robot de manera inalámbrica a través de Bluetooth, sirviendo como interfaz para el envío de comandos de movimiento al robot.

Finalmente, se recopila toda la información, se documenta el desarrollo del proyecto y se elabora la memoria necesaria para la defensa del TFG ante el tribunal.

1.3 Estructura de la memoria

Este documento está estructurado en varios capítulos, cada uno diseñado para abordar diferentes aspectos del desarrollo del proyecto de un robot hexápodo. A continuación, se presenta una visión general de la estructura del trabajo:

Capítulo 1. Introducción:

Ofrece una visión general del proyecto, incluyendo la motivación detrás de su desarrollo, los objetivos específicos y la metodología empleada. Detalla las razones que motivaron la elección del proyecto, enfatizando la importancia y relevancia de los robots hexápodos en la robótica moderna, así como la inspiración tomada de la naturaleza y su aplicabilidad en el diseño robótico. También se describe la estructura del documento.

Capítulo 2. Estado del Arte:

Proporciona un análisis exhaustivo de los avances actuales en el campo de los robots hexápodos. Incluye una revisión de la literatura y los proyectos existentes, destacando las ventajas, aplicaciones y tendencias recientes en la robótica hexápoda.

Capítulo 3 Fundamentos de los Hexápodos:

Se introducen conceptos de los hexápodos como robots paralelos, diferentes tipos de modos de locomoción, estabilidad y morfología de estos tipos de robots.

Capítulo 4. Hardware:

Describe el diseño y desarrollo de la estructura mecánica y los componentes electrónicos del robot. Se detalla el proceso de diseño de las piezas mediante software CAD, la impresión 3D y el ensamblaje, así como la elección y configuración del hardware electrónico, incluyendo el microcontrolador ESP32 y los módulos PWM.

Capítulo 5. Software y algoritmos matemáticos:

Explica el desarrollo de los algoritmos de control y la programación del microcontrolador. Se incluyen los cálculos de trayectorias y cinemática inversa, la implementación de scripts en Python y el desarrollo de la aplicación móvil para controlar el robot.

Capítulo 6. Resultados:

Presenta los resultados obtenidos durante el desarrollo y las pruebas del robot. Se incluyen las pruebas de funcionamiento, el análisis de los resultados y una discusión sobre las limitaciones y posibles mejoras futuras del prototipo.

Capítulo 7. Presupuesto:

Detalla los costos asociados al desarrollo del proyecto, desglosando los gastos en componentes mecánicos, electrónicos, software utilizado y mano de obra.

Capítulo 8. Conclusiones:

Resume las conclusiones generales del proyecto y ofrece recomendaciones para futuras investigaciones y mejoras en el robot hexápodo. Se discuten las posibles aplicaciones y desarrollos adicionales que podrían expandir las capacidades del robot.

Capítulo 9. Referencias Bibliográficas

Lista de todas las fuentes consultadas durante el desarrollo del proyecto, proporcionando una base sólida de conocimientos en robótica hexápoda.

Capítulo 10. Anexos:

Incluye información adicional relevante que complementa el contenido principal del proyecto. Esto puede incluir diagramas y planos del diseño, el código fuente de los scripts y el manual de usuario de la aplicación móvil.

2 Estado del Arte

En este capítulo se abordará el análisis de la situación actual de los robots hexápodos, sus ventajas, la inspiración biológica detrás de su diseño y las aplicaciones prácticas y tecnológicas que están emergiendo en este campo, así como prototipos de hexápodos reales.

2.1 Ventajas de los robots hexápodos

En el diseño de robots móviles, existe una gran discusión entre el uso de ruedas y el uso de miembros articulados. Aunque es más fácil brindar movilidad a un robot con ruedas, los robots articulados ofrecen beneficios significativos. Entre estos beneficios se destaca la capacidad de desplazarse sobre una variedad de terrenos, sin importar cuán escarpados o resbalosos sean. Esta movilidad superior, aunque a una velocidad máxima menor, los hace ideales para situaciones de rescate o misiones de exploración donde los seres humanos no pueden aproximarse.

La discusión entre robots con patas se centra en cuadrúpedos y hexápodos. Mientras que los cuadrúpedos pueden moverse a mayores velocidades debido a sus movimientos repetitivos y fluidos, los hexápodos ofrecen ventajas notables a pesar de ser más lentos:

- Cambio de Orientación con Facilidad: Los hexápodos pueden cambiar su orientación con mayor facilidad.
- Adaptación a Terrenos Rugosos: Son capaces de lidiar con terrenos más rugosos y escalar con mayor facilidad.
- Diversidad de Modos de Caminar: Pueden utilizar diferentes tipos de trotes según el terreno, como el modo alternante (caminar típico), modo cuadrúpedo (mayor velocidad) y gateo (más lento, pero con mayor estabilidad).

Además, los movimientos de los hexápodos son más fáciles de estudiar, ya que los insectos en los que se basan estos movimientos tienen redes neuronales simples, facilitando el diseño de algoritmos de movimiento bio-inspirados [1].

2.2 Inspiración biológica

El diseño y la funcionalidad de los robots hexápodos se inspiran en gran medida en la biología de los insectos y otros animales de múltiples patas. La naturaleza ha perfeccionado a lo largo de millones de años mecanismos de locomoción altamente adaptables, y los ingenieros han recurrido a estos modelos naturales para desarrollar robots con capacidades similares.

La cinemática de las patas de los insectos también proporciona una rica fuente de inspiración para los algoritmos de control de los robots hexápodos. Al estudiar cómo los insectos coordinan sus patas para caminar, los ingenieros pueden desarrollar algoritmos

de control bio-inspirados que permiten a los robots hexápodos moverse de manera eficiente y adaptativa.

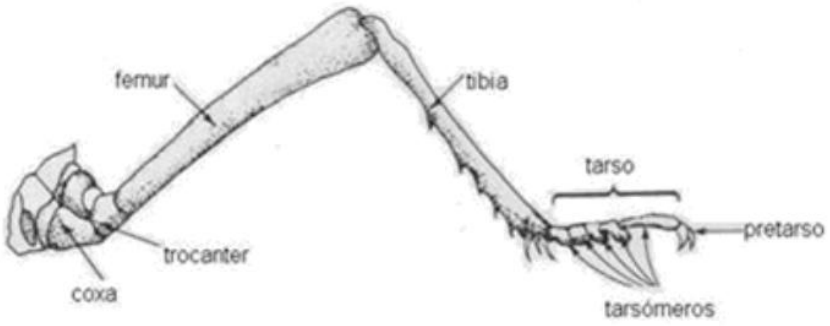


Figura 1. Nomenclatura de una pata. Fuente: [2]

La Figura 5 muestra la estructura típica de una pata de insecto, la cual consta de varios segmentos: coxa, trocánter, fémur, tibia, tarso.... Cada segmento juega un papel clave en la locomoción del insecto, proporcionando movilidad y estabilidad. Los robots hexápodos utilizan una estructura similar para lograr movimientos complejos y adaptativos [2].

En el prototipo desarrollado en este proyecto, se ha seguido esta inspiración biológica utilizando piezas impresas en 3D y motores para construir estructuras equivalentes a la coxa, el fémur y la tibia de los insectos.

2.3 Hexápodos en la Actualidad

En la actualidad, los robots hexápodos encuentran aplicaciones en una variedad de campos debido a sus capacidades únicas de movilidad y estabilidad. A continuación, se describen algunos de los escenarios en los que estos robots están siendo utilizados:

2.3.1 Aplicaciones de los robots Hexápodos

2.3.1.1 Exploración Planetaria

Los hexápodos son considerados para misiones de exploración en terrenos extraterrestres, como la superficie de Marte. Su capacidad para adaptarse a terrenos irregulares y su estabilidad superior los hacen ideales para navegar en ambientes hostiles y desconocidos. Estos robots pueden enfrentar obstáculos y operar en condiciones extremas, lo que es esencial para misiones espaciales [3].

2.3.1.2 Rescate y Búsqueda

En situaciones de desastres naturales, los robots hexápodos pueden despeñar una tarea determinante en misiones de búsqueda y rescate.

Gracias a su habilidad para superar obstáculos y acceder a áreas que serían inaccesibles para robots con ruedas, pueden explorar escombros y estructuras colapsadas para encontrar sobrevivientes.

Esta capacidad es vital en escenarios donde la rapidez y la capacidad de maniobra son fundamentales [4].

Figura 2.Robot hexápodo para ubicaciones remotas y entornos hostiles. Fuente: [5]

2.3.1.3 Industria y Automatización

En el ámbito industrial, los hexápodos se utilizan en aplicaciones que requieren movimientos precisos y controlados. Por ejemplo, se emplean en la fabricación de semiconductores, ensamblaje de componentes electrónicos y en la industria automotriz para tareas que demandan alta precisión y repetibilidad. Los sistemas de control de movimiento de hexápodos permiten ajustar el centro de rotación y realizar alineaciones avanzadas con facilidad, mejorando así la eficiencia y precisión en los procesos de manufactura [3] [4].

Figura 3. Sistema de posicionamiento hexápodo (Plataforma Stewart). Fuente: [6]

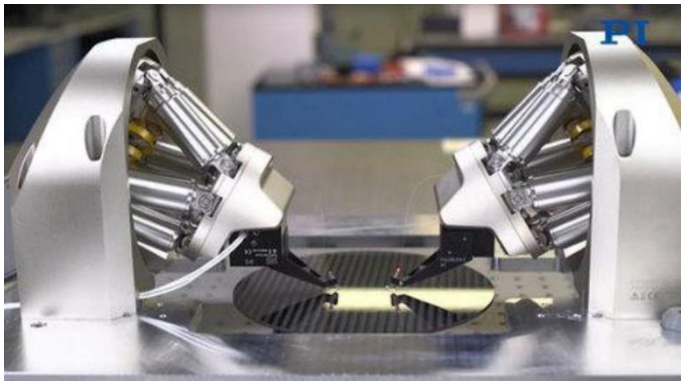


Figura 4. Hexápodo probando semiconductores y componentes electrónicos. Fuente: [5]

2.3.1.4 Exploración Submarina

Los hexápodos también se están adaptando para la exploración submarina. Su capacidad para moverse y adaptarse a superficies irregulares es ventajosa en ambientes acuáticos, donde pueden realizar investigaciones en el lecho marino y en áreas de difícil acceso para vehículos submarinos tradicionales. Esta flexibilidad y capacidad de adaptación los convierte en herramientas valiosas para estudios oceanográficos y misiones de exploración submarina [4].

2.3.2 Prototipos de hexápodos reales

En la actualidad, existen varios prototipos de robots hexápodos que han sido desarrollados con distintos fines, desde la investigación académica hasta aplicaciones industriales y de rescate. A continuación, se describen algunos de los prototipos más destacados y sus características.

2.3.2.1 RHex

RHex es un robot hexápodo desarrollado por Boston Dynamics, inspirado en la biología y diseñado específicamente para desplazarse en terrenos difíciles. Con su diseño robusto y modular, RHex es capaz de moverse con agilidad sobre una variedad de superficies complicadas, como rocas, barro, arena, nieve y vías de tren.

Las especificaciones técnicas de RHex son impresionantes. Con un ancho de 39 centímetros, una altura de 20 centímetros y una longitud de 57 centímetros, este robot tiene unas dimensiones compactas. El peso del RHex estándar es de 12,5 kilogramos, mientras que la versión X-RHex pesa 8,6 kilogramos. A pesar de su tamaño, RHex puede alcanzar una velocidad máxima de 9,72 kilómetros por hora, lo que lo hace extremadamente eficiente en misiones que requieren rapidez y movilidad.

El robot está equipado con una serie de sensores avanzados, incluyendo cámaras, giroscopios, acelerómetros y otros sensores opcionales que miden la posición, la corriente y la temperatura de los motores. Estos sensores permiten a RHex adaptarse y responder adecuadamente a su entorno. Además, cuenta con seis motores de corriente continua que le proporcionan los seis grados de libertad necesarios para su movilidad.

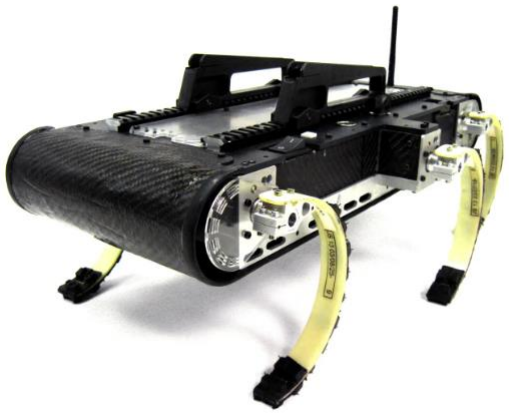


Figura 5. Robot hexápodo RHex. Fuente: [7]

Construido con materiales duraderos como el aluminio y la fibra de carbono, RHex es resistente al agua y al polvo. Su capacidad de cálculo está respaldada por una computadora principal basada en Intel, con la opción de añadir una computadora de carga útil según las necesidades de la misión. El software del robot funciona con el sistema operativo Linux, y el control de este puede programarse en lenguajes como C, Python o Matlab.

La fuente de energía de RHex proviene de dos baterías de polímero de litio de 144 Wh, que le proporcionan una autonomía de hasta seis horas de funcionamiento continuo. Desde su introducción en 2001, el proyecto RHex ha estado en constante desarrollo, adaptándose a nuevas tecnologías y requisitos de misión [7].

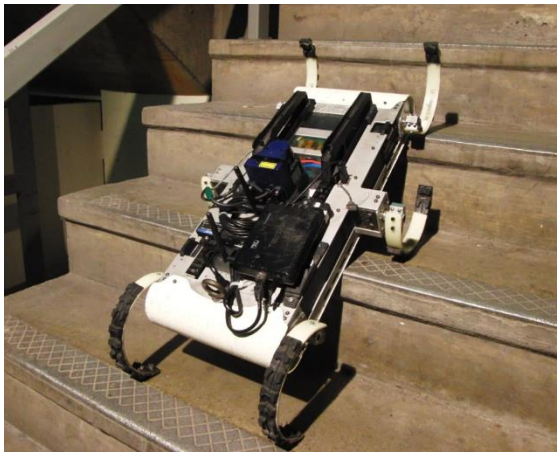


Figura 7. RHex subiendo por escaleras. Fuente: [7]

Figura 6. RHex caminando sobre arena. Fuente: [7]

Figura 8. RHex subiendo una colina rocosa y empinada en medio del desierto. Fuente: [7]

2.3.2.2 LAURON V

LAURON V es un robot hexápodo desarrollado por el Centro de Investigación de Tecnología de la Información (FZI) en Alemania, en el año 2013. Este robot, inspirado biológicamente, está diseñado para operar en áreas donde los sistemas con ruedas tendrían dificultades, ofreciendo una solución eficiente para tareas de monitoreo del entorno y manipulación móvil.

Con un ancho de 80 centímetros, una altura de 61 centímetros (84 cm con las piernas completamente extendidas) y una longitud de 90 centímetros, LAURON V presenta dimensiones que le permiten maniobrar con facilidad en terrenos complicados. Su peso es de 43,5 kilogramos y puede alcanzar una velocidad de hasta 0,5 kilómetros por hora.

Figura 9. Robot hexápodo LAURON V. Fuente: [8]

Este robot está equipado con avanzados sensores, incluyendo cámaras térmicas, cámaras de imagen 3D y sensores láser 3D personalizados, que le permiten detectar objetos, mapear su entorno y planificar rutas. Cada pata de LAURON V tiene cuatro articulaciones con codificadores rotatorios Avago HEDS9000, asegurando un movimiento preciso y controlado.

LAURON V cuenta con 28 motores distribuidos en sus patas y otras partes del cuerpo, permitiéndole adaptarse dinámicamente al terreno y manejar resbalones, evitando caídas. Los materiales de construcción incluyen aluminio de calidad aeroespacial para las patas y componentes impresos en 3D con PLA y PET, proporcionando una estructura robusta y ligera.

El sistema de control de LAURON V está basado en un ordenador principal con una placa Mini ITX y un procesador Intel Core i7, acompañado de placas adicionales para tareas específicas como control de alta precisión, iluminación y análisis láser 3D. Además, utiliza controladores Arduino y Raspberry Pi para diversas funciones.

El software de LAURON V corre en Ubuntu 14.04 con ROS Indigo para el control de bajo nivel, mientras que el control de alto nivel se realiza en el marco PLEXNAV, que soporta planificación basada en OMPL y reconocimiento de objetos.

LAURON V obtiene su energía de dos paquetes de baterías de polímero de litio de 24 V y 80 Wh, con la opción de añadir más baterías para extender su autonomía durante las misiones. El costo aproximado de este avanzado robot es de 150.000 euros.

Inspirado en el insecto palo común “Carausius morosus”, sus patas y patrones de marcha están optimizados matemáticamente para emular los movimientos de este insecto [8].

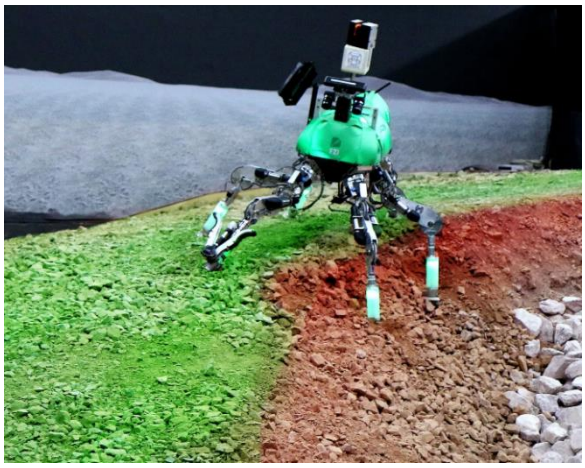


Figura 11. LAURON V atravesando superficies irregulares. Fuente: [8]

Figura 10. LAURON V en una misión simulada a Marte. Fuente: [8]

2.3.2.3 DASH

DASH es un robot hexápodo de pequeño tamaño, inspirado en los insectos, desarrollado en 2009 por la Universidad de California en Berkeley. Creado por Paul Birkmeyer y el profesor Ronald Fearing en el Laboratorio de Milisistemas Biomiméticos, DASH está diseñado para aplicaciones de búsqueda y rescate, así como para la detección remota. Su nombre, "Hexápodo Dinámico Autónomo Disperso", refleja su capacidad para moverse con agilidad y autonomía en diversos entornos.

Con unas dimensiones de 10 centímetros de ancho, 5 centímetros de alto y 10 centímetros de largo, y un peso de apenas 16 gramos, DASH es extremadamente ligero y compacto. A pesar de su tamaño, puede alcanzar una velocidad de 5,4 kilómetros por hora, equivalente a 1,5 metros por segundo, lo que es 15 veces su longitud corporal por segundo. Este robot es capaz de correr, trepar obstáculos que superan su propia altura y sobrevivir a caídas desde cualquier altura.

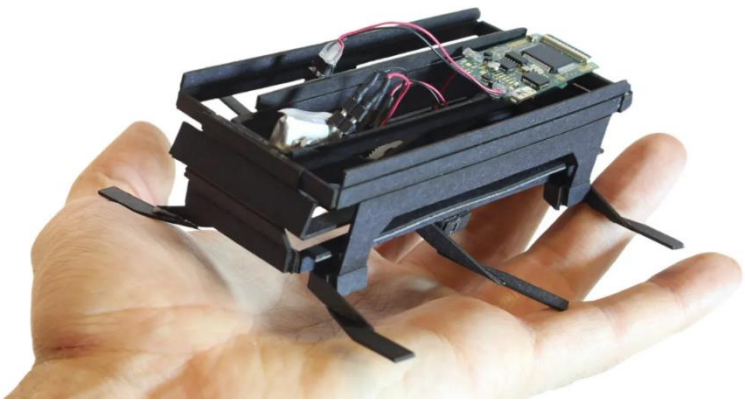


Figura 12. Robot hexápodo DASH. Fuente: [9]

DASH está equipado con un acelerómetro de tres ejes y un giroscopio de tres ejes para medir su orientación y movimiento. Además, puede llevar una cámara de móvil opcional para tareas específicas de detección y monitoreo. Utiliza un motor de corriente continua con escobillas para mover sus patas y un servomotor de aleación con memoria de forma para girar. Estos actuadores permiten que DASH se adapte a su entorno, trepando y navegando sobre obstáculos con facilidad.

El cuerpo de DASH está compuesto por papel con juntas de flexión de polímero, lo que le proporciona una estructura ligera pero sorprendentemente robusta. Su capacidad de cálculo está gestionada por un microcontrolador de 40 MHz y su funcionamiento está controlado por un firmware personalizado. La fuente de energía de DASH es una batería de polímero de litio de 3,7 V y 50 mAh, que le da una autonomía de entre 30 minutos y varias horas, dependiendo del uso.

El desarrollo de DASH incluyó numerosas revisiones y mejoras, utilizando materiales económicos y procesos de fabricación eficientes. Aunque ya no está en producción activa, DASH ha sido reemplazado por CLASH, otro pequeño hexápodo que se basa en las lecciones aprendidas de DASH para ofrecer una plataforma robótica trepadora aún más avanzada.

DASH es ideal para aplicaciones de búsqueda y rescate debido a su capacidad para correr rápidamente, trepar obstáculos y sobrevivir a caídas. Su diseño flexible y robusto lo convierte en una herramienta valiosa para la detección remota y la exploración en terrenos difíciles. Su costo de fabricación es relativamente bajo, aproximadamente 100 dólares, lo que lo hace accesible para diversas investigaciones y aplicaciones prácticas [9].

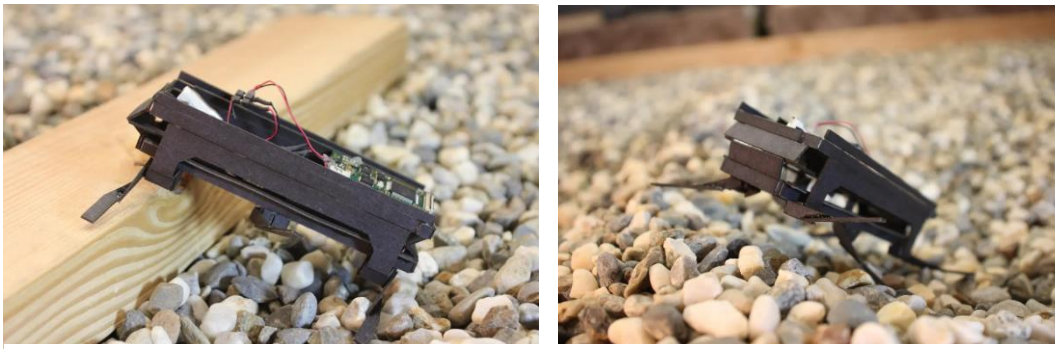


Figura 13. Izquierda: DASH trepando un obstáculo más alto que él. Derecha: DASH caminando a través de rocas. Fuente: [9]

2.3.2.4 Genghis

Genghis es un robot hexápodo desarrollado en 1989 por Rodney Brooks en el Instituto Tecnológico de Massachusetts (MIT). Este robot fue concebido para demostrar cómo comportamientos complejos, como el gateo, pueden surgir a partir de una red de controladores simples.

Con unas dimensiones de 25 centímetros de ancho y 35 centímetros de largo, y un peso de 1 kilogramo, Genghis es un robot compacto pero avanzado para su época. Equipado con 12 sensores de fuerza, seis sensores infrarrojos, dos inclinómetros y dos bigotes sensibles al tacto, este robot tiene la capacidad de percibir y reaccionar a su entorno de manera efectiva. Para moverse, Genghis utilizaba 12 servomotores controlables por posición, lo que le proporcionaba 12 grados de libertad y le permitía adaptarse a diferentes superficies y obstáculos.

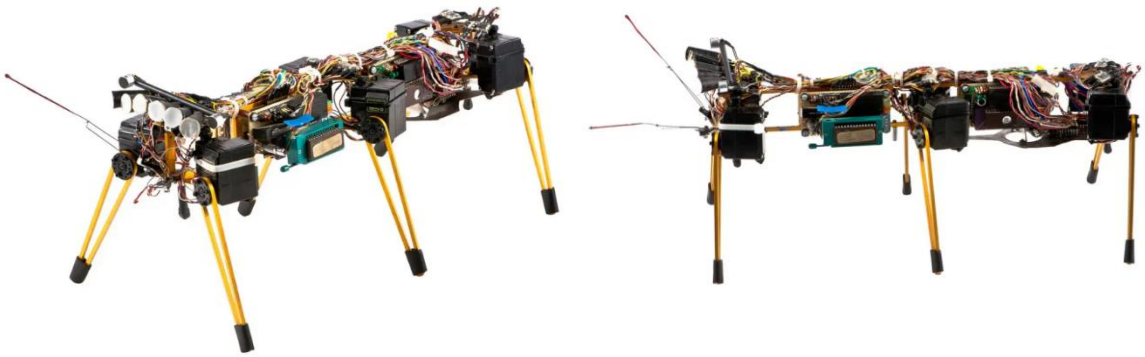


Figura 14. Robot hexápodo Genghis. Fuente: [10]

La energía del robot provenía de tres pilas de plata y zinc, y su sistema de control se basaba en cuatro microprocesadores de 8 bits integrados. La memoria total del sistema era de 1 KB de RAM y 10 KB de EPROM. Genghis empleaba un enfoque innovador de control conocido como "arquitectura de subsunción", que vinculaba la percepción limitada y específica de la tarea directamente con la acción, sin necesidad de un sistema de control centralizado.

El desarrollo de Genghis comenzó en 1988, en respuesta a un taller del Laboratorio de Propulsión a Chorro de la NASA sobre micro naves espaciales. Brooks utilizó el robot como banco de pruebas para explorar su concepto de la arquitectura de subsunción. Su idea era que el comportamiento complejo, como gatear y trepar obstáculos, no requería un sistema de control central, sino que podía surgir de controladores simples y distribuidos.

Basado en su experiencia con Genghis, Brooks propuso que la exploración del sistema solar debería basarse en misiones económicas y rápidas, utilizando un gran número de robots autónomos producidos en masa en lugar de naves espaciales más complejas y costosas. Esta visión fue descrita por Brooks y Anita M. Flynn en un artículo de 1989 titulado "Rápido, barato y fuera de control: una invasión robótica del sistema solar". Genghis finalmente fue exhibido en el Museo Nacional del Aire y el Espacio del Instituto Smithsonian en Washington, DC.

Aunque el desarrollo de Genghis fue interrumpido, su legado en el campo de la robótica es significativo. El robot demostró cómo sistemas simples y modulares podían dar lugar a comportamientos complejos y eficientes, influenciando el diseño de futuros robots y facilitando el avance de la tecnología robótica [10].

2.3.2.5 Daisy

Daisy es un kit de robótica hexápoda desarrollado por Robótica HEBI en 2017, pensado para investigadores que estudian la locomoción y el control del movimiento de robots. Este sistema destaca por su capacidad de personalización y expansión, permitiendo a los usuarios ajustarlo según las necesidades específicas de cada proyecto de investigación.

Con unas dimensiones de 100 centímetros de ancho, 50 centímetros de alto y 100 centímetros de largo, y un peso de 18 kilogramos, Daisy es un robot robusto y versátil. Puede alcanzar una velocidad de 1 kilómetro por hora. Equipado con avanzados sensores, Daisy cuenta con detección propioceptiva en sus articulaciones, sensores de posición, velocidad y par, así como una unidad de medición inercial (IMU). Además, dispone de sensores internos de temperatura, corriente y voltaje, lo que le permite monitorizar su funcionamiento en tiempo real.

Figura 15. Robot hexápodo Daisy. Fuente: [11]

El robot utiliza doce actuadores X8-9 y seis actuadores X8-16, proporcionando un total de 18 grados de libertad, con 3 grados de libertad por cada una de sus seis patas. Su estructura está construida principalmente con tubos de aluminio y soportes mecanizados, y cuenta con cubiertas de cables impresas en 3D, lo que le otorga una excelente durabilidad y ligereza.

En cuanto a su capacidad de cálculo, Daisy se basa en una computadora Intel NUC en chasis y microprocesadores ARM Cortex M4 en cada articulación. La computadora puede configurarse con Ubuntu o Windows. El software del kit incluye código de demostración y API completas disponibles en C++, ROS, Python y MATLAB, facilitando así el desarrollo y la integración en diversos proyectos de investigación.

Daisy se alimenta con dos baterías LiGo de 98 Wh de Grin Technologies, que son intercambiables en caliente y proporcionan una autonomía de entre 1 a 2 horas, dependiendo del uso. El costo aproximado del kit es de 85,000 dólares.

Los creadores de Daisy han añadido actuadores adicionales para dotar al robot de un brazo con 6 grados de libertad y una pinza, ampliando así su funcionalidad. Este diseño modular permite a los investigadores ajustar y expandir el sistema según sus necesidades

específicas, haciendo de Daisy una herramienta extremadamente versátil en el ámbito de la robótica y la investigación de la locomoción [11].

Figura 16. Daisy con brazo manipulador. Fuente: [11]

3 Fundamentos de los Hexápodos

Este capítulo abordará conceptos esenciales sobre los hexápodos, un tipo de robot articulado que forma parte de la familia de los robots paralelos, para facilitar la comprensión en los capítulos siguientes. Se explorarán los diferentes modos de locomoción o “walking gaits”, las diversas morfologías y una explicación detallada de lo que constituye un robot paralelo.

3.1 ¿Qué es un robot paralelo?

Los robots paralelos son una clase de robots en los cuales el efector final está soportado por múltiples brazos, todos conectados a una base común. Estos robots se diferencian de los robots en serie, donde cada articulación sigue a la anterior en una única cadena de movimiento. Los robots paralelos son conocidos por su alta rigidez, precisión y capacidad de carga, lo que los hace adecuados para diversas aplicaciones avanzadas [12] [13].

3.1.1 Tipos de robots paralelos

Los robots paralelos se pueden clasificar en varios tipos, cada uno con sus características específicas y aplicaciones destacadas. A continuación, se describen los principales tipos de robots paralelos.

3.1.1.2 Robot Delta

El robot Delta es uno de los más conocidos y utilizados en aplicaciones industriales. Fue desarrollado inicialmente para operaciones de alta velocidad y baja carga, como el empaquetado, ensamblaje y manipulación de alimentos. Sus características principales son una estructura de tres brazos paralelos que soportan el efector final, permitiendo alta velocidad y precisión en movimientos pick-and-place. Este tipo de robot se utiliza principalmente en la industria alimentaria, farmacéutica y en el ensamblaje de piezas pequeñas [14].

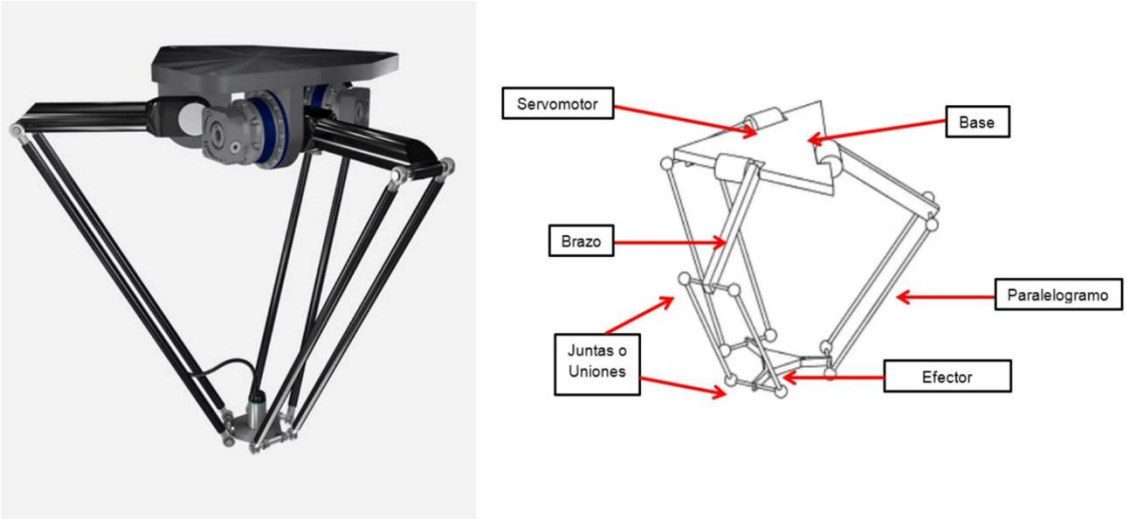


Figura 17. Izquierda: Robot delta. Derecha: Partes de un robot delta. Fuente: [14]

3.1.2.2 Robot Gough-Stewart (Plataforma Stewart)

La plataforma Stewart, también conocida como robot Gough-Stewart, es un tipo de robot paralelo de seis grados de libertad, ampliamente utilizado en simuladores de vuelo y movimientos de precisión en aplicaciones industriales y de investigación.

Su estructura consiste en seis actuadores lineales conectados a una plataforma móvil, lo que permite movimientos precisos y controlados en seis grados de libertad (tres traslaciones y tres rotaciones). Las aplicaciones típicas de este robot incluyen simuladores de movimiento, máquinas herramienta y medicina quirúrgica. [15]

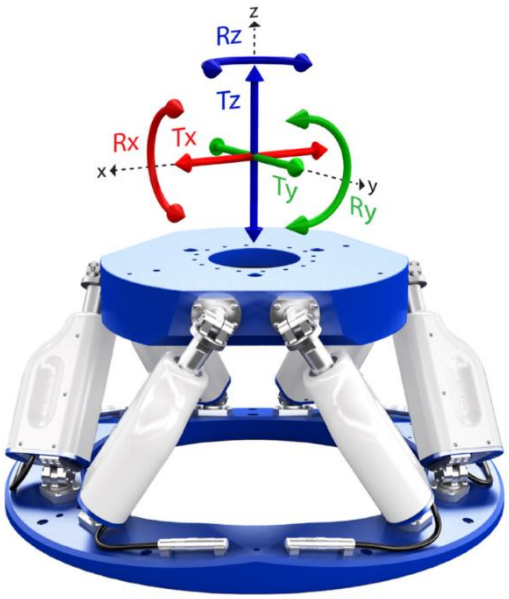


Figura 18. Plataforma Stewart y la representación de sus grados de libertad. Fuente: [15]

1.1.2.3 Robot Hexápodo

El robot hexápodo es un tipo de robot paralelo caracterizado por tener seis patas articuladas que soportan y controlan el efector final. Cada una de las patas proporciona un grado de libertad, permitiendo al robot realizar movimientos precisos y complejos en seis grados de libertad. Este diseño le otorga una gran estabilidad y capacidad para maniobrar en terrenos irregulares, lo que lo hace ideal para aplicaciones en exploración espacial e inspección de superficies [16].

Figura 20. Hexapod Phoenix. Fuente: [17]

Figura 19. Hexapod PhantomX. Fuente: [18]

3.1.2 Ventajas de los robots paralelos frente a los robots serie

Los robots paralelos ofrecen varias ventajas significativas en comparación con los robots de tipo serie, lo que los hace especialmente útiles en aplicaciones industriales exigentes.

Una de las principales ventajas de los robots paralelos es su mayor rigidez y precisión. Debido a su diseño estructural, que distribuye las cargas a través de múltiples cadenas cinemáticas independientes, los robots paralelos son más rígidos. Esta rigidez se traduce en una mayor precisión en el posicionamiento y en la ejecución de tareas que requieren exactitud.

Otra ventaja importante es la relación optimizada entre masa y carga. Los robots paralelos pueden soportar cargas más pesadas en comparación con su propia masa, permitiendo mayores aceleraciones y velocidades durante el movimiento. Esto es especialmente beneficioso en aplicaciones donde se requieren movimientos rápidos y precisos, como en las líneas de ensamblaje.

Otra ventaja clave es la mayor velocidad de operación. La rigidez estructural y la optimización de la relación masa-carga permiten que los robots paralelos operen a velocidades más altas sin comprometer la precisión. Esto es ideal para tareas que requieren rapidez y eficiencia [19].

3.2 Modos de locomoción

3.2.1 Fases del paso en un hexápodo

El desplazamiento del hexápodo se logra mediante la ejecución de pasos, cada uno de los cuales se divide en dos etapas como se muestra en la figura 21. La primera etapa es el "swing" o balanceo de la pata en el aire hacia su posición final. Desde esta posición, comienza la segunda etapa denominada "stance" o postura de apoyo, donde la pata se desplaza por el suelo desde su posición final hasta la inicial. En este documento, nos referiremos a esta etapa como "arrastre". La ejecución cíclica de estas dos etapas a lo largo de una trayectoria y con tiempos sincronizados entre las patas genera la variedad de modos de locomoción disponibles [20].

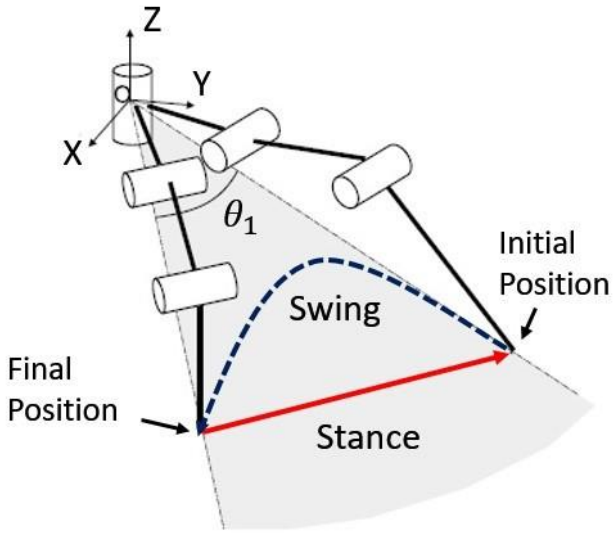


Figura 21. Fases de Balanceo y arrastre del hexápodo. Fuente: [20]

3.2.2 Marcha en trípode o “Tripod Gait”.

Se caracteriza por el empleo de tres de las seis patas para moverse a la vez, formando dos trípodes alternantes. Este patrón es común en insectos y otros hexápodos por su estabilidad y eficiencia.

La marcha en trípode constara pues de dos fases.

1. Fase de apoyo de patas impares: En esta fase las patas delantera derecha (1), media izquierda (5) y trasera derecha (3) están en contacto con el suelo y sostienen e impulsan el cuerpo del animal, mientras que las otras tres patas se balancean y mueven hacia delante.
2. Fase de apoyo de patas pares: Las patas que estaban realizando el arrastre se levantan y realizan un balanceo hacia el punto final, mientras que las patas pares realizan ahora el arrastre.

La principal ventaja de esta marcha es su buen equilibrio entre estabilidad y velocidad. Es estable debido a que siempre forma tres puntos de contacto con el suelo manteniendo el centro de gravedad en todo momento entre el triángulo formado. Y aunque no es la marcha más rápida posible, es uno de los más seguros y versátiles para moverse en terrenos irregulares [21] [22].

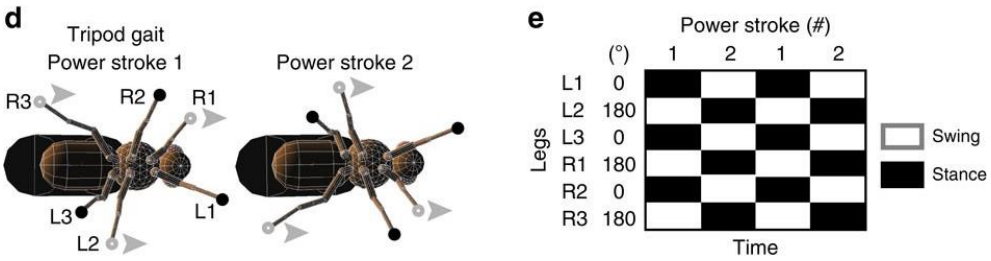


Figura 22.Marcha en trípode. Fuente: [22]

3.2.2 Marcha en forma de ola o "Wave Gait".

Se trata de la marcha más estable al asegurar siempre 5 puntos de apoyo, pero en contra parte es la más lenta en términos de velocidad de desplazamiento.

Recibe el nombre por la secuencia de movimientos similar a una onda que se propaga alrededor del cuerpo. Por ejemplo, en un hexápodo con patas numeradas del 1 al 6, las patas se moverían de manera individual siguiendo una secuencia de movimiento 1-2-3-4-5-6, repitiéndose este ciclo a lo largo del desplazamiento [23].

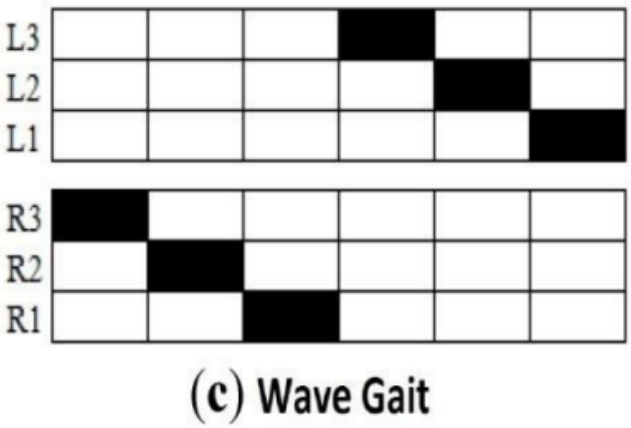


Figura 23. Patrón de pisadas Marcha ondulatoria, Negro: balanceo / Blanco arrastre, Fuente: [23]

3.2.3 Marcha ondulante o “Ripple Gait”

La locomoción mediante paso ondulante en un robot hexápodo se caracteriza por ser más lenta que la marcha de trípode, pero más rápida que la marcha en forma de ola, en contra parte proporciona mayor estabilidad que la marcha en trípode, pero menor que la marcha en forma de ola. Este tipo de marcha sigue un patrón cíclico donde dos patas se balancean simultáneamente para avanzar, mientras que las otras cuatro permanecen en contacto con el suelo para propulsar al robot. La clave de su estabilidad radica en asegurar que las dos patas en fase de balanceo no se encuentren en el mismo lado del robot. Este patrón permite mantener siempre al menos cuatro patas en contacto con el suelo, garantizando así mayor estabilidad durante el desplazamiento [23] [20].

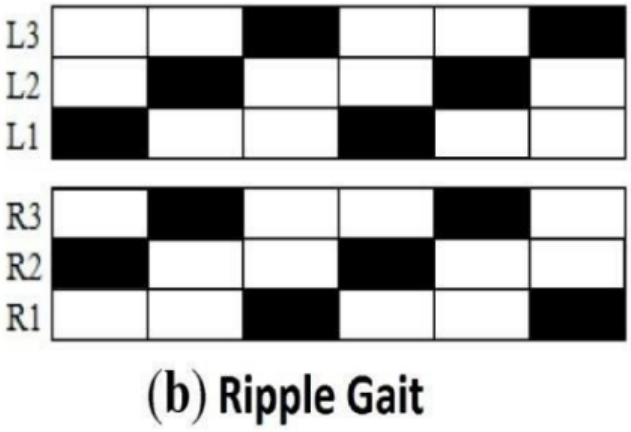


Figura 24. Marcha ondulante, Negro: balanceo / Blanco arrastre. Fuente: [23]

3.3 Estabilidad del robot

La estabilidad es una característica primordial en el diseño de robots hexápodos. Una diferencia fundamental entre los mecanismos de locomoción es si son estática o dinámicamente estables. Un robot estáticamente estable debe mantenerse estable durante todo su ciclo de marcha, sin necesidad de ningún tipo de fuerza para equilibrarlo.

Mientras el robot está estáticamente estable, la proyección vertical de su centro de masa debe estar ubicada dentro del polígono de soporte, que se forma entre sus puntos de apoyo. Cuando el centro de masas se ubica fuera del polígono de soporte, el robot caerá, a menos que sea dinámicamente estable y mantenga el equilibrio durante su marcha debido a la inercia, siendo estáticamente inestable una vez detenido el movimiento.

Por ejemplo, los pies de un robot cuadrúpedo abarcan un rectángulo. Si el robot levanta una pata, este rectángulo se convierte en un triángulo. Si la proyección del centro de masa del robot está fuera de este triángulo, el robot caerá. Un robot dinámicamente estable puede superar este problema cambiando su configuración rápidamente para evitar la caída.

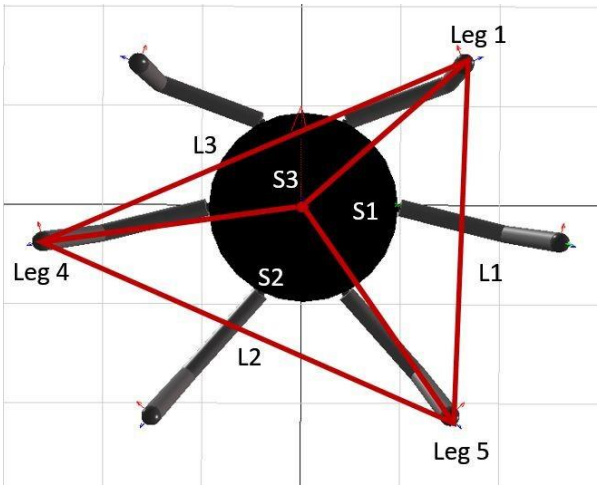


Figura 26.Ejemplo de robot estáticamente estable. Hexápodo con 3 puntos de apoyo. Fuente: [20]

Figura 25.Ejemplo de robot dinámicamente estable. Segway RMP. Fuente: [25]

3.4 Morfologías y distribuciones de las patas

Al diseñar el cuerpo de un hexápodo, las morfologías y la distribución de las patas juegan un papel crucial, determinado en gran medida por el propósito funcional del hexápodo. A continuación, exploramos diferentes diseños morfológicos y distribuciones y sus características principales.

3.4.1 Morfología de la pata

Podemos diferenciar 2 familias de patas más comunes: patas biarticuladas y tri-articuladas. Las patas biarticuladas sacrifican la articulación tibia-fémur, con las implicaciones negativas en cuanto a flexibilidad de movimiento y omnidireccionalidad. Sin embargo, son más fáciles de programar y requieren menos servomotores, lo que también se traduce en una electrónica más sencilla y un bucle de control menos complejo.

Por otro lado, las patas tri-articuladas proporcionan un movimiento más natural y fluido, capaz de realizar una mayor gama de movimientos. Claro está que esto añade complejidad al desarrollo del robot.

Es posible encontrar patas con un número diferente de articulaciones, aunque es menos común. Por ejemplo, el RHex [7] que cuenta con una sola articulación por pata que la hace rotar a modo de “rueda” para desplazarse.

3.4.2 Distribución de las patas

La distribución de las patas determinará la forma del cuerpo, aunque las formas que puede adoptar el cuerpo de un hexápodo son diversas en general encontramos 3 distribuciones comunes:

1. Distribución en rectangular o paralelo.

Las patas están distribuidas en dos líneas paralelas a los laterales del robot. Esta disposición es ideal para robots que se enfocan en la velocidad lineal de movimiento. La simetría implica que todas las patas siguen las mismas trayectorias durante el avance, lo que simplifica el desarrollo del software. Sin embargo, esta configuración dificulta el giro y el movimiento omnidireccional del hexápodo.

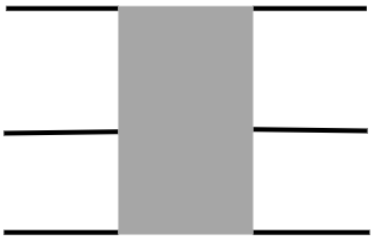


Figura 27. Hexápodo con distribución rectangular. Fuente: Elaboración propia

2. Distribución hexagonal o circular.

Esta distribución se caracteriza por su omnidireccionalidad, ya que las patas están dispuestas equidistantemente, formando una circunferencia alrededor del cuerpo del robot. Esto implica que todas las direcciones son igualmente favorables para el movimiento, pero también significa que ninguna dirección se destaca en particular. Debido a la disposición más dispersa de las patas en sentido de avance, el desplazamiento tiende a ser más lento en comparación con un hexápodo con distribución lineal, especialmente cuando se utiliza el mismo hardware. Para compensar esto, las patas en esta configuración deben dar pasos más largos y pueden requerir servomotores con mayor torque para mantener un rendimiento equivalente al de un hexápodo con distribución lineal en términos de velocidad.

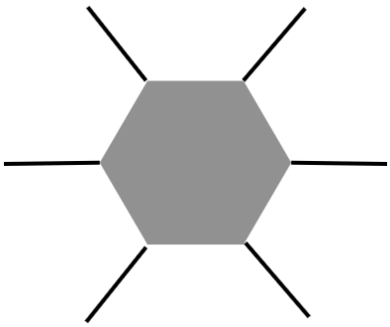


Figura 28. Hexápodo con distribución hexagonal. Fuente: Elaboración propia

3. Distribución octogonal y derivados.

Por último, existen distribuciones octogonales y similares que están en un término medio entre las distribuciones rectangulares y hexagonales. Constan de dos hileras de patas claramente separadas como la distribución rectangular sin embargo las patas de los extremos se encuentran ligeramente rotadas, lo que favorece la capacidad omnidireccional del hexápodo sin comprometer en gran medida su avance lineal.

El prototipo que se describe en este documento pertenece a esta subcategoría de hexápodos.

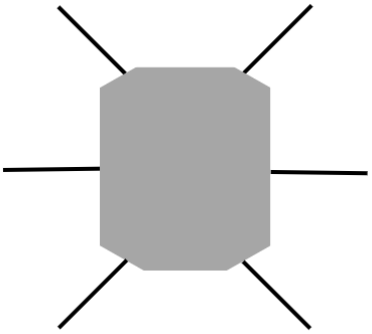


Figura 29. Hexápodo con distribución octogonal. Fuente: Elaboración propia

3.5 Ventajas de los hexápodos frente a los a otros vehículos convencionales.

Una de las principales ventajas de los robots hexápodos es su capacidad de movimiento omnidireccional, lo que les otorga una flexibilidad y adaptabilidad superior en comparación con otros tipos de robots. Esta capacidad se debe a la disposición de sus seis patas, que les permite moverse en cualquier dirección con gran precisión.

Los robots hexápodos pueden cambiar de dirección fácilmente y maniobrar en espacios reducidos, lo que es importante para tareas en entornos complicados y dinámicos. Esta movilidad superior es especialmente útil en situaciones donde el terreno es irregular, escarpado o resbaladizo, ya que las patas del robot pueden ajustarse para mantener el equilibrio y proporcionar tracción en cualquier dirección.

A diferencia de los robots con ruedas, que están limitados por su capacidad de girar y moverse solo en direcciones específicas, los robots hexápodos pueden desplazarse lateralmente, girar sobre su eje y moverse en diagonal. Esta capacidad de movimiento omnidireccional los hace ideales para misiones de rescate y exploración, donde es necesario navegar por terrenos impredecibles y cambiantes [1].

Los robots hexápodos tienen la capacidad de moverse en cualquier dirección y pueden emplear diversos modos de locomoción para adaptarse a las características del terreno y optimizar su estabilidad y eficiencia energética.

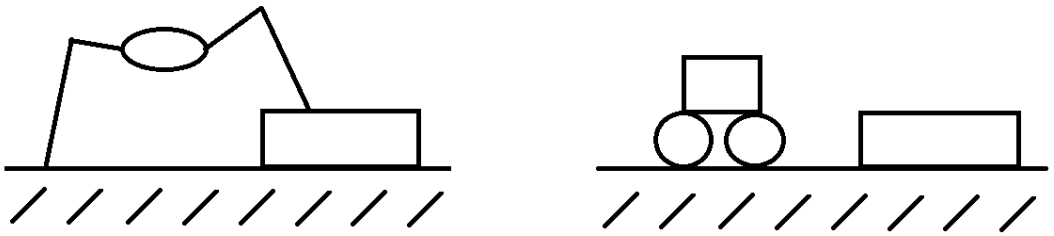


Figura 30. Izquierda: Capacidad de superar obstáculos de un robot hexápodo. Derecha: Dificultades de un robot con ruedas para superar obstáculos. Fuente: Elaboración propia.

Figura 31. Izquierda: Estabilidad de un robot hexápodo en terreno irregular. Derecha: Inestabilidad de un robot con ruedas en terreno irregular. Fuente: Elaboración propia.

Aunque el prototipo de hexápodo desarrollado en este proyecto solo puede avanzar y retroceder en líneas rectas, y rotar sobre sí mismo para cambiar de dirección usando un modo alternante de caminar, estas capacidades pueden ser expandidas en futuras iteraciones mediante la planificación de nuevas trayectorias empleando los algoritmos desarrollados. Estas limitaciones no desmerecen las ventajas inherentes de los robots hexápodos, cuya diversidad de modos de locomoción demuestra su potencial para adaptarse a una amplia gama de terrenos y situaciones. Su diseño y estructura permiten implementar en el futuro capacidades de movimiento más complejas, aprovechando su flexibilidad y adaptabilidad natural, lo que destaca sus ventajas en comparación con otros tipos de robots.

Aunque la estabilidad dinámica es deseable para movimientos ágiles y de alta velocidad, los robots deben diseñarse para poder cambiar fácilmente a una configuración estáticamente estable. Un ejemplo de esto es un corredor cuadrúpedo, que puede tener configuraciones tanto estáticamente como dinámicamente estables. Cuando corre, siempre tiene dos patas en el aire y alterna entre ellas más rápido de lo que el robot podría caer en cualquier dirección. [26]

En el caso de los robots hexápodos, su diseño les permite lograr una mayor estabilidad estática y dinámica. Aunque es posible caminar de manera estáticamente estable con solo

cuatro patas, la mayoría de los animales y robots prefieren seis patas para mayor estabilidad. Seis patas permiten al animal o robot mover tres patas a la vez mientras las otras tres mantienen una postura estable.

En el prototipo de hexápodo desarrollado en este proyecto, se ha priorizado la estabilidad estática debido a su capacidad para avanzar y retroceder en líneas rectas y rotar sobre sí mismo. Aunque este prototipo no implementa todas las capacidades dinámicas posibles, su diseño de seis patas proporciona una base sólida para futuras mejoras. Las tres patas en contacto constante con el suelo garantizan que el robot mantenga su equilibrio incluso en terrenos irregulares.

4 Hardware

En este capítulo se abordará el diseño y desarrollo del hardware del robot hexápodo. Se detallarán los componentes mecánicos y electrónicos utilizados, el proceso de fabricación de las piezas, y la integración de todos los elementos para crear un prototipo funcional.

4.1 Hardware mecánico

En este apartado se hablará del diseño y desarrollo de los componentes mecánicos del robot hexápodo, incluyendo la estructura y las piezas móviles. Se detallarán los materiales utilizados, las herramientas empleadas para la fabricación y los procedimientos de ensamblaje.

4.1.1 Diseño de la estructura

El diseño de la estructura mecánica del robot hexápodo se realizó utilizando el software Fusion 360. Este software permitió crear un modelo detallado de cada componente del robot, asegurando que todas las piezas encajaran perfectamente y funcionaran de manera coordinada. Las piezas principales del robot, como la coxa, el fémur y la tibia, se diseñaron inspiradas en la anatomía de las patas de los insectos.

En este apartado se detallarán las características del prototipo final del robot hexápodo. Es importante mencionar que, durante el desarrollo del proyecto, se realizaron diferentes diseños y componentes hasta llegar a la versión final. Estos cambios fueron necesarios para la correcta funcionalidad, estabilidad y fiabilidad del robot. A continuación, se describe el proceso de diseño y desarrollo de los componentes finales.

4.1.1.1 Diseño de componentes específicos

4.1.1.1.1 Coxa

La coxa actúa como la base de la pata, conectándola al cuerpo del robot. Se diseñó para ser robusta y ligera, permitiendo una conexión firme con el chasis del robot y proporcionando una base estable para el movimiento de las patas. La coxa es una caja en la cual se inserta el servomotor del "codo" y que permite sujetarlo al cuerpo mediante unos orificios traseros. Se atornilla con tornillos M3 de 15mm y tuercas, fijando firmemente el servomotor.

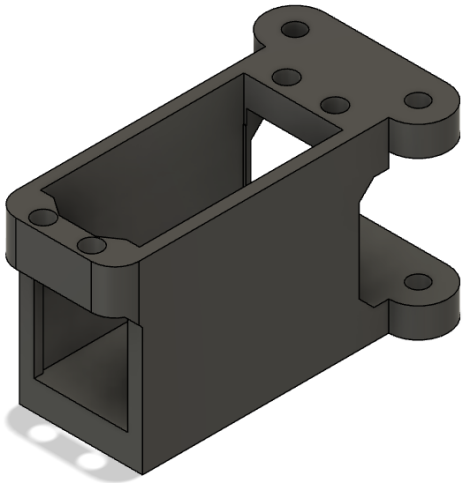


Figura 33. Caja (Coxa). Fuente: Elaboración propia.

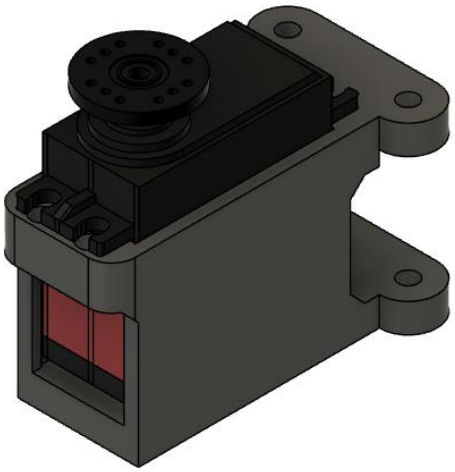


Figura 32. Caja (Coxa) con servomotor. Fuente: Elaboración propia.

4.1.1.1.2 Fémur

El fémur consta de dos piezas: una en forma de T que sostiene los servos y otra superior que actúa como tapa para los servomotores. En el fémur se encuentran los otros dos servomotores, uno que permite la elevación de la pierna (el servomotor más cercano al codo) y otro que permite el movimiento de la tibia (el servomotor más alejado de la pata con respecto a la coxa). Estas partes se ensamblan mediante tornillos M3 de 50mm, tuercas y arandelas, proporcionando una estructura robusta y estable.

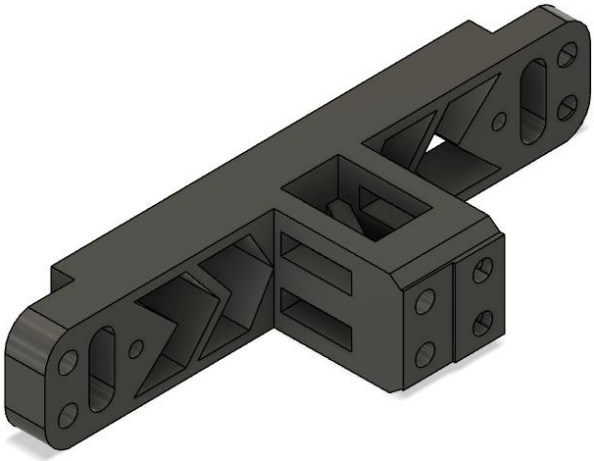


Figura 35. Fémur (parte superior). Fuente: Elaboración propia.

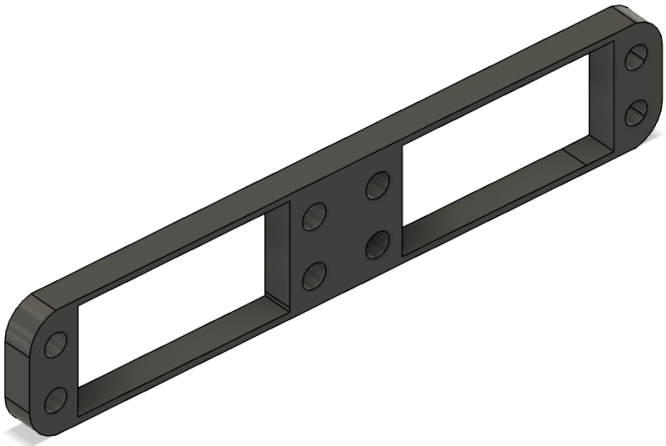


Figura 34. Fémur (parte superior). Fuente: Elaboración propia.

Para optimizar el peso del fémur, se identificaron zonas de la pieza que no contribuían significativamente a la resistencia estructural y se ahuecaron. Este enfoque permitió reducir el peso del fémur sin comprometer su integridad estructural.

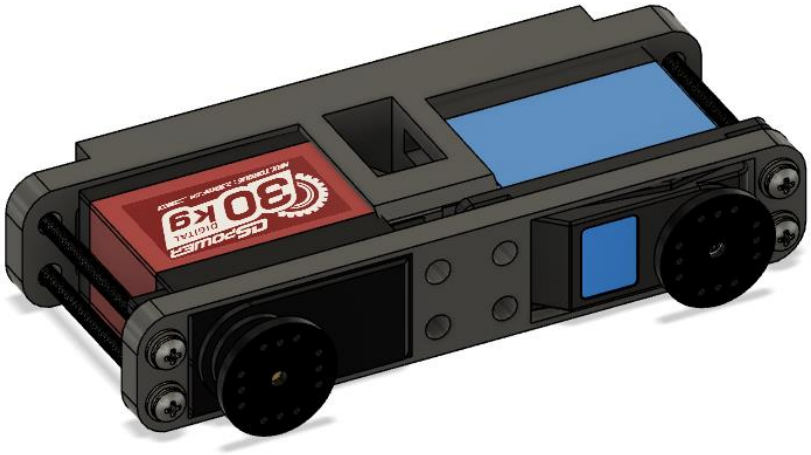


Figura 36. Fémur con servomotores ensamblado. Fuente: Elaboración propia.

4.1.1.1.3 Tibia

La tibia se diseñó para ser recta y lo más corta posible para ganar en estabilidad y firmeza. Este diseño ayuda a reducir el torque necesario para realizar las operaciones de movimiento. La tibia se conecta al fémur mediante una pieza en forma de U, que permite el movimiento articulado. Al igual que el fémur, se optimizó la tibia haciéndola hueca en partes no críticas, lo que permitió disminuir su peso manteniendo la resistencia necesaria para soportar las cargas del movimiento.

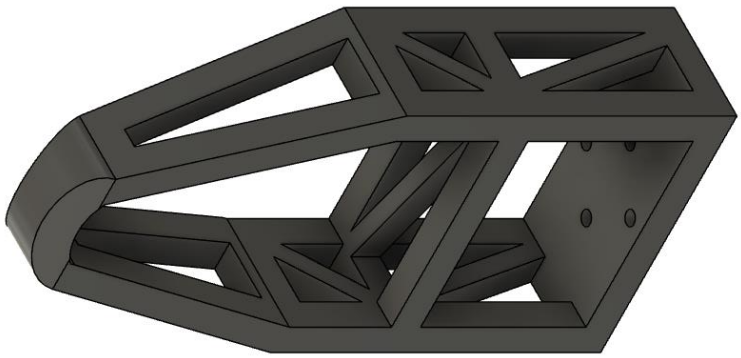


Figura 37. Tibia. Fuente: Elaboración propia.

4.1.1.1.4 Articulaciones

Para simular las articulaciones y unir la coxa con el fémur y la tibia con el fémur, se utilizaron piezas en forma de U que conectan las partes de manera perpendicular. Estas piezas permiten el movimiento en las articulaciones y se sujetan con tornillos M3 de 15mm y 20mm. Se ensamblan a la cabeza circular del servomotor con tornillos M2 de 8mm.

1. Articulación Coxa-Fémur: Se usaron piezas en forma de U de 68mm. La articulación se sujeta mediante tornillos ciegos, que actúa de bisagra, asegurando que la conexión quede firme, pero permita el movimiento. Los agujeros para los tornillos se hicieron de manera que el hueco presente tanto en la coxa y como en el fémur sea menor que el hueco presente en la U. Esto asegura que la articulación

esté sujeta por la coxa o el fémur en cada caso, pero que quede libre por parte de la U, permitiendo así que la articulación pueda girar. Para asegurar las piezas se utilizaron tornillos M2 de 8mm que se fijan en el attachment circular del servomotor.

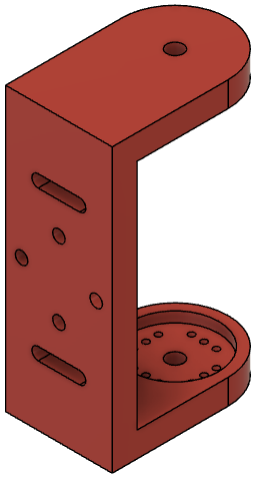


Figura 39. U 68. Fuente: Elaboración propia.

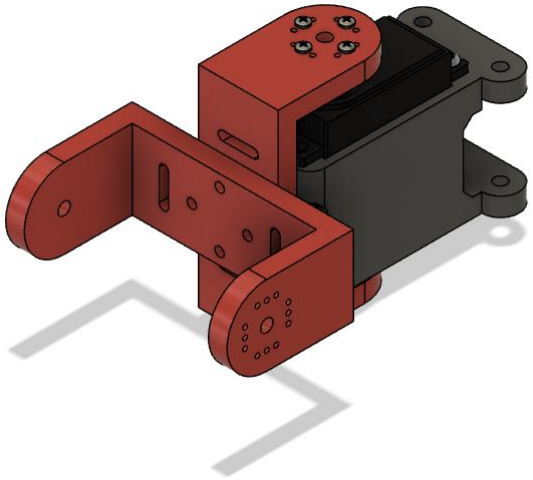


Figura 38. Articulación coxa-fémur ensamblada. Fuente: Elaboración propia

2. Articulación Fémur-Tibia: Se usaron piezas en forma de U de 71.4mm, conectadas de manera similar a la articulación coxa-fémur, permitiendo el movimiento articulado y asegurando la estabilidad. Esta unión también se realizó con tornillos ciegos para permitir el movimiento libre de la articulación.

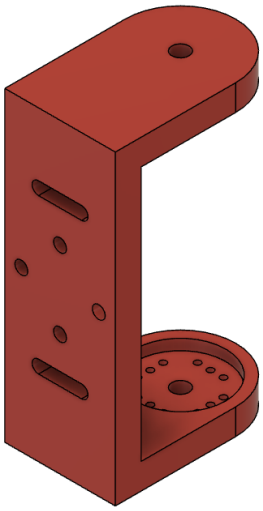


Figura 40. U 71.4. Fuente: Elaboración propia.

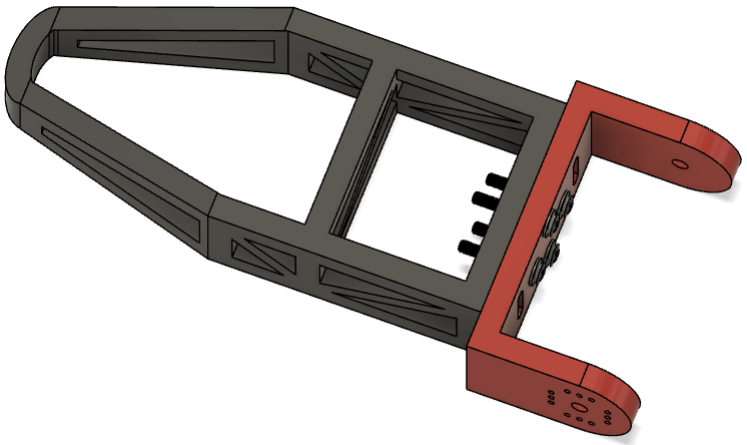


Figura 41. Articulación fémur-tibia ensamblada. Fuente: Elaboración propia.

4.1.1.1.5 Cuerpo

El cuerpo del robot está compuesto por dos piezas planas en forma octogonal con vías para tornillería dispuestas de manera equidistante. Dos de estos huecos están en paralelo y los otros cuatro a intervalos de 45º, permitiendo ensamblar las patas al cuerpo con la orientación elegida.

Las patas se ensamblan al cuerpo usando tornillos M3 de 20mm para la parte inferior y tornillos M3 de 40mm para la superior, utilizando tuercas y arandelas para asegurar las uniones.

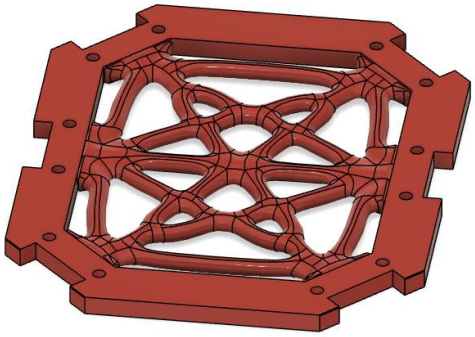


Figura 43. Cuerpo (parte inferior). Fuente: Elaboración propia.

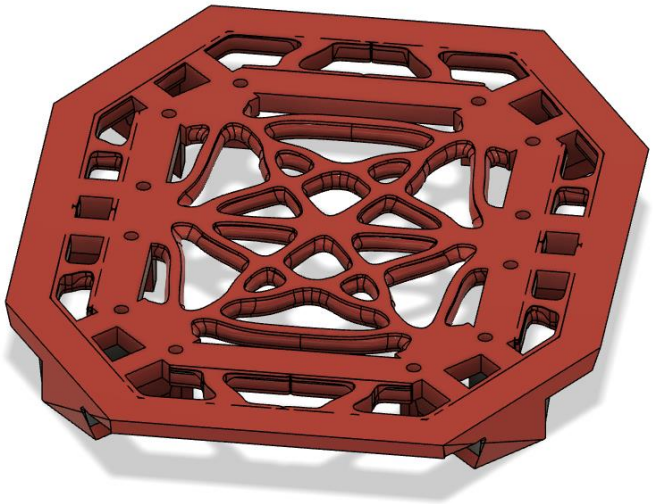


Figura 42. Cuerpo (parte superior). Fuente: Elaboración propia.

4.1.1.1.6 Pata

Para el diseño de las patas se optó por una morfología triarticulada para maximizar la estabilidad y adaptabilidad. Se estudiaron diferentes configuraciones y proporciones y se optó por una disposición que permitiera un movimiento fluido y estable sin altas demandas de torque.

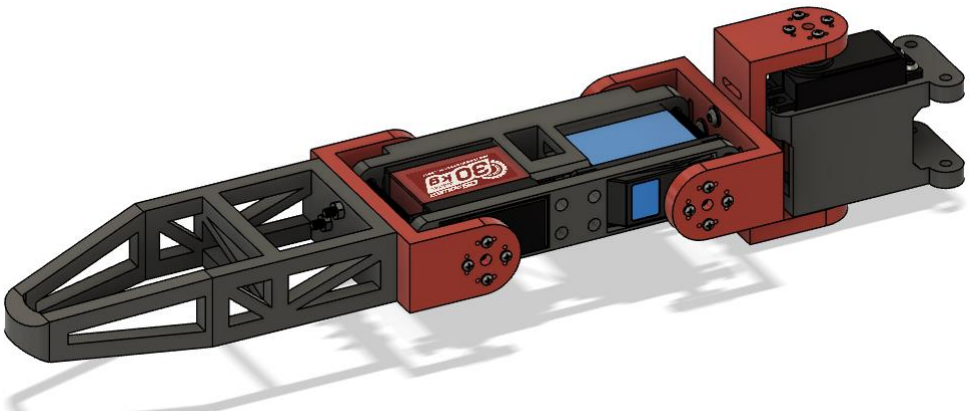


Figura 44. Pata ensamblada. Fuente: Elaboración propia.

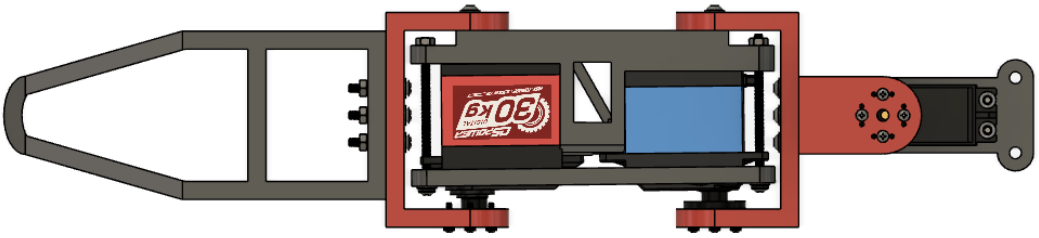


Figura 45. Pata vista desde arriba. Fuente: Elaboración propia.

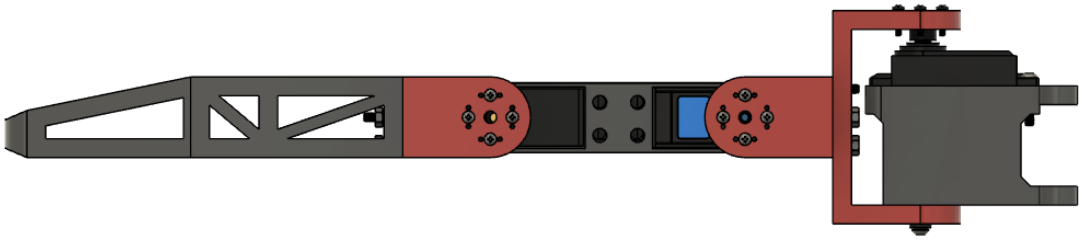


Figura 46. Pata vista desde un lateral. Fuente: Elaboración propia.

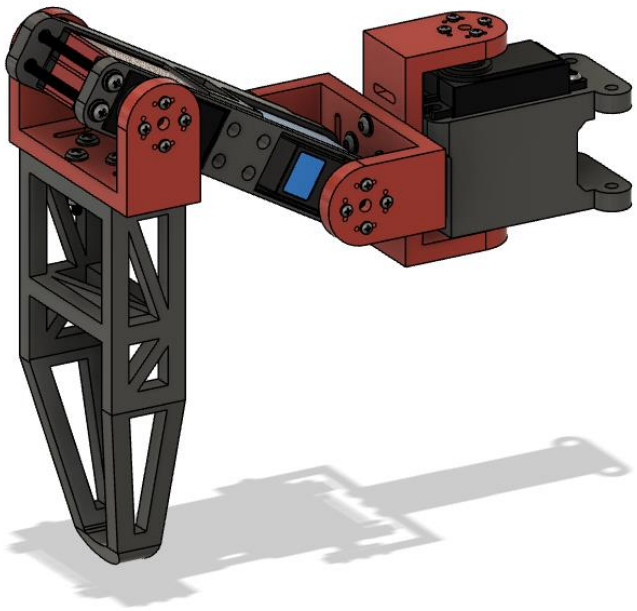


Figura 47. Pata en posición de reposo. Fuente: Elaboración propia.

- Integración de los Servomotores

Los servomotores se integraron en el diseño de manera que estuvieran protegidos y no interfirieran con el movimiento de las patas. El diseño está pensado para que los mismos componentes sean soportes para los servomotores, asegurando que estos estuvieran firmemente sujetos y alineados correctamente con las articulaciones de las patas. Además, se consideró la facilidad de acceso a los servomotores para su mantenimiento y calibración.

- Innovaciones y Ajustes

Se realizaron numerosos ajustes de dimensiones y piezas hasta lograr el prototipo deseado. Al ensamblarlo todo en Fusion 360, se verificó que las piezas encajaran correctamente. Sin embargo, al llevarlo a la realidad, las impresiones mostraron imprecisiones de algunos milímetros, lo que condujo a realizar varios cambios en las impresiones y la elección de tornillería.

- Comparación entre el Diseño inicial y el prototipo final

Como se mencionó anteriormente en el desarrollo del robot hexápodo, se realizaron diferentes iteraciones de diseño para optimizar la funcionalidad, estabilidad y requisitos de torque del robot. A continuación, se presentan imágenes del diseño inicial y se comparan con el prototipo final, destacando las mejoras implementadas en el nuevo diseño.

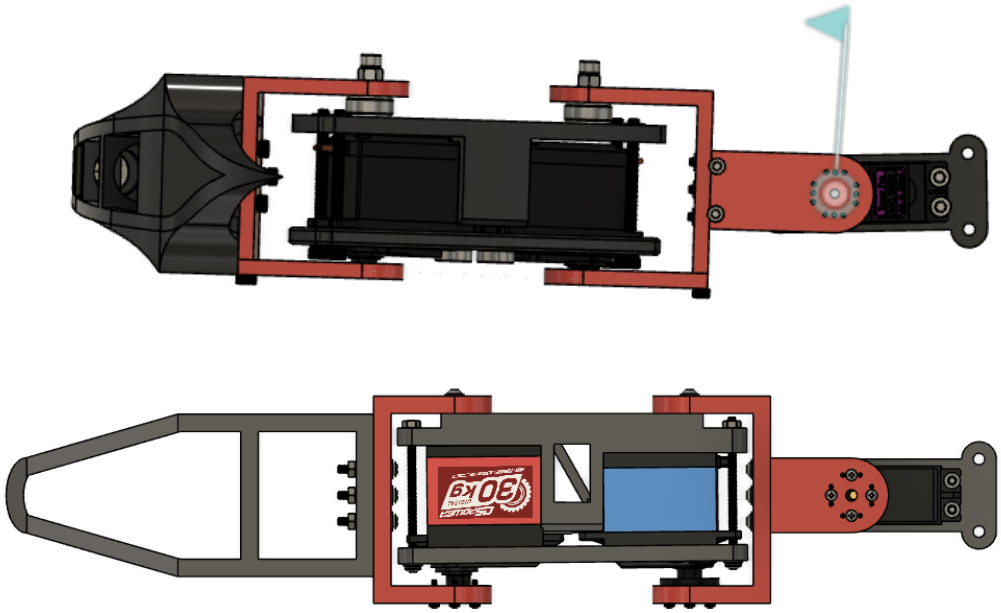


Figura 48. Arriba: Pata del diseño inicial vista desde arriba. Abajo: Pata del prototipo final vista desde arriba. Fuente: Elaboración propia.

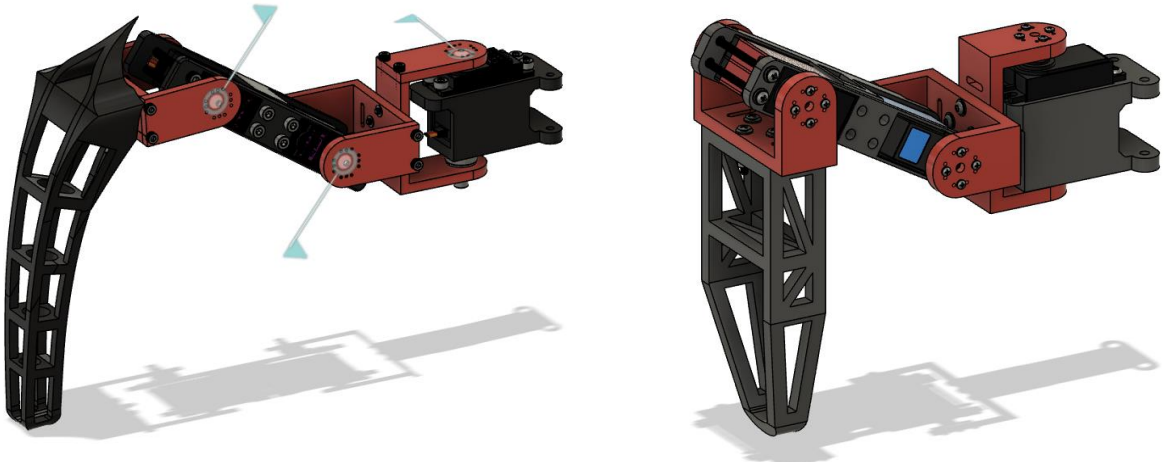


Figura 49. Izquierda: Pata del diseño inicial en posición de reposo. Derecha: Pata del prototipo final en posición de reposo. Fuente: Elaboración propia.

Diferencias y Mejoras:

1. Tibia

La tibia en el diseño inicial era más larga y menos optimizada en términos de peso y torque. En el prototipo final, la tibia se diseñó para ser recta y lo más corta posible, mejorando significativamente la estabilidad y firmeza del robot. Además, se hicieron huecos en zonas no críticas para reducir el peso sin comprometer la resistencia estructural.

2. Espacio entre Articulaciones y Componentes

El espacio entre las articulaciones y los componentes era mayor, lo que resultaba en una pata más larga que demandaba mayor torque. Se optimizó el espacio entre las articulaciones y los componentes, logrando una longitud de pata más corta. Esta optimización mejoró la robustez del robot y también redujo el torque necesario para realizar las operaciones de movimiento.

3. Tornillería

La tornillería usada en el diseño inicial no era adecuada y resultaba en pizas dañadas o mal sujetas. En el prototipo final, se mejoró la selección de tornillería, utilizando tornillos M3 de 15mm y 20mm, así como tornillos M2 de 8mm, tuercas y arandelas para asegurar los componentes de manera más eficiente y ligera.

4. Piezas en Forma de U

Las piezas en forma de U que se usaban para las articulaciones estaban compuestas por dos piezas distintas, una L y un parte plana, que se ensamblaban para formar una U, lo que complicaba el ensamblaje y comprometían la robustez del diseño. En el nuevo diseño, las piezas en forma de U se componen de una sola pieza, lo que facilita el ensamblaje y mejora la robustez de las articulaciones.

El prototipo final del robot hexápodo muestra mejoras significativas respecto al diseño inicial. Estas modificaciones han resultado en un robot más estable y ligero, capaz de realizar movimientos precisos con mayor facilidad y fiabilidad.

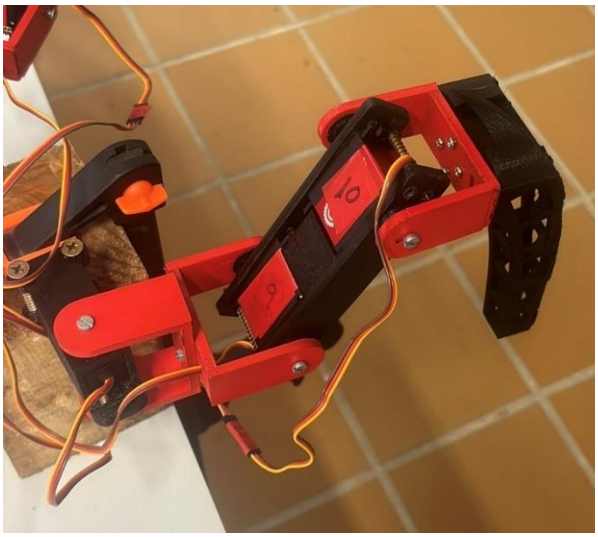


Figura 50. Pata del diseño inicial en el laboratorio I. Fuente: Elaboración propia.

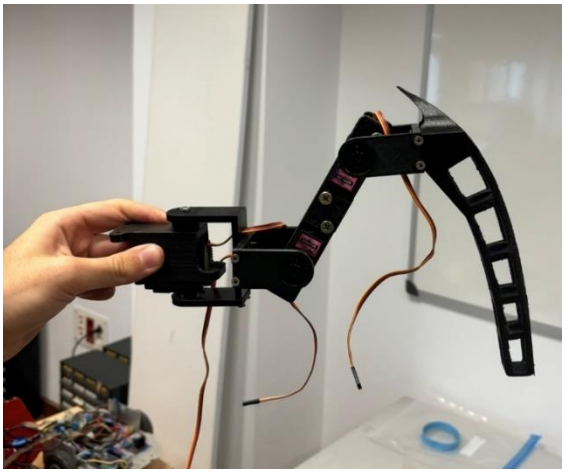


Figura 51. Pata del diseño inicial en el laboratorio II. Fuente: Elaboración propia.

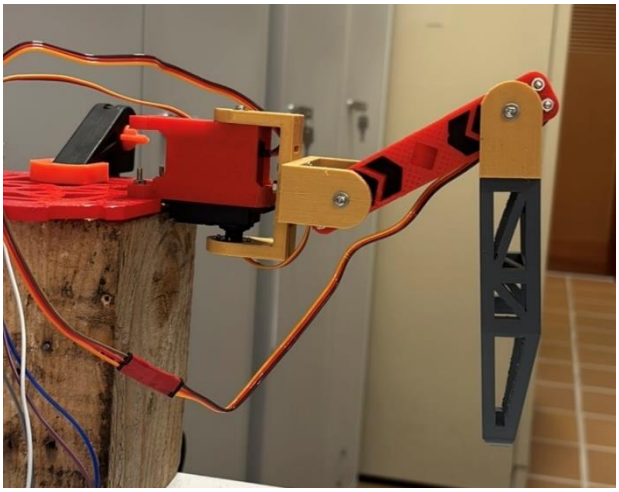


Figura 52. Pata del prototipo final en el laboratorio I.
Fuente: Elaboración propia.

Figura 53. Pata del prototipo final en el laboratorio II.
Fuente: Elaboración propia.

Figura 58. Hexapod del diseño inicial. Fuente: Elaboración propia.

Figura 59. Hexapod del prototipo final. Fuente: Elaboración propia.

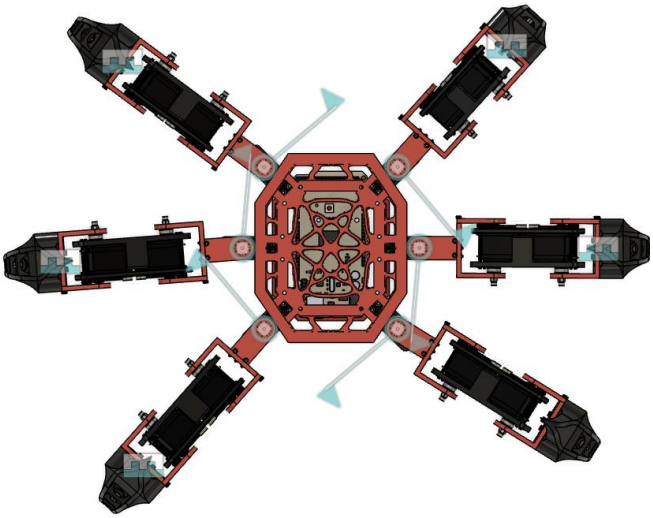


Figura 57. Hexapod del diseño inicial visto desde arriba. Fuente: Elaboración propia.

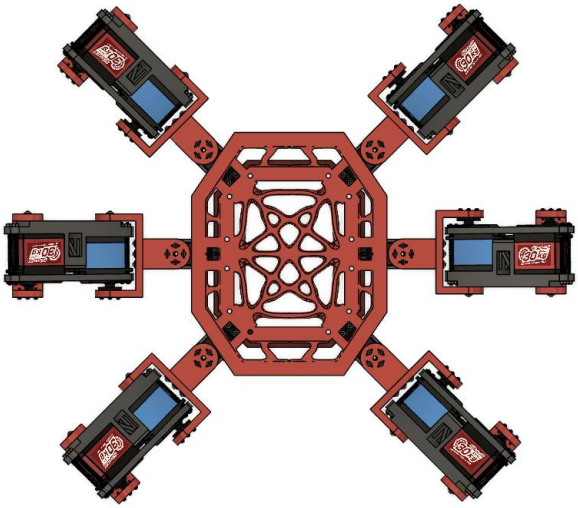


Figura 56. Hexapod del prototipo final visto desde arriba. Fuente: Elaboración propia.

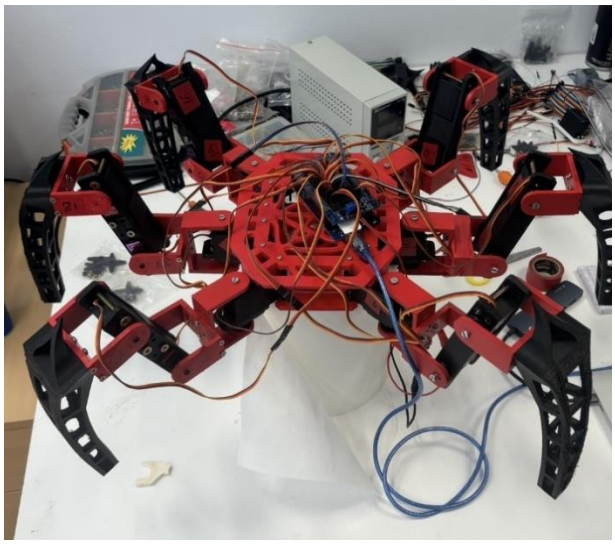


Figura 55. Hexapod del diseño inicial en el laboratorio. Fuente: Elaboración propia.

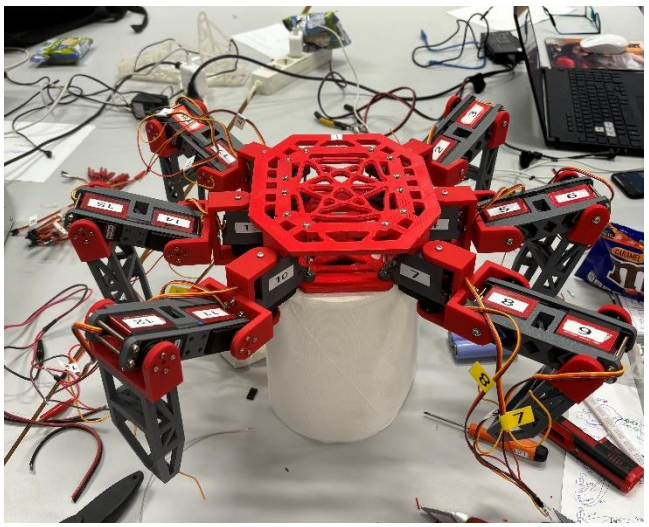


Figura 54. Hexapod del prototipo final en el laboratorio. Fuente: Elaboración propia.

4.1.2 Impresión 3D y ensamblaje

En esta sección se detalla el proceso de impresión 3D y el ensamblaje de las piezas del robot hexápodo, describiendo los parámetros de impresión, los materiales utilizados, los problemas encontrados y las soluciones implementadas, así como los pasos específicos del ensamblaje.

4.1.2.1 Proceso de impresión 3D

Para la impresión de las piezas se usaron varias impresoras facilitadas por la universidad y por nuestro tutor. Algunas piezas también se mandaron a imprimir en tiendas especializadas debido a limitaciones de tiempo. A continuación, se detallan los parámetros de impresión utilizados para las diferentes impresoras:

Parámetro	Creality CR-10S	BQ Witbox 2	Hephestos	Prusa
Grosor de la pared	0.8 mm	1.2 mm	0.8 mm	0.8 mm
Densidad de relleno	20%	20%	20%	20%
Patrón de relleno	Giroide	Giroide	Giroide	Giroide
Temperatura de impresión	210ºC	245ºC	215ºC	210ºC
Temperatura de la placa de impresión	60ºC	60ºC	60ºC	60ºC
Velocidad de impresión	75mm/s	60mm/s	60mm/s	65mm/s

Tabla 1. Parámetros de impresión usados para las distintas impresoras 3D.

El parámetro de altura de capa común para todas las impresoras e impresiones fue de 0.2mm.

4.1.2.2 Filamento utilizado

Para la fabricación de las piezas del robot hexápodo se utilizó filamento PLA (ácido profiláctico), debido a sus múltiples ventajas. El PLA es conocido por su baja contracción, lo que minimiza el riesgo de deformaciones durante la impresión.

Esta característica es significativa para garantizar la precisión de las piezas impresas. Además, el PLA se adhiere bien a la cama de impresión y es compatible con la mayoría de las impresoras 3D disponibles en el mercado, lo que facilita el proceso de impresión.

Otro punto a favor del PLA es su respeto por el medio ambiente, ya que es biodegradable y se fabrica a partir de recursos renovables. También ofrece una buena resistencia y durabilidad, adecuándose perfectamente a la fabricación de piezas mecánicas que requieren cierta rigidez y resistencia.

Es por eso por lo que, en el campo de la robótica, el PLA se utiliza para fabricar prototipos y componentes estructurales de robots debido a su facilidad de impresión y buenos acabados superficiales. Es adecuado para aplicaciones de corta vida útil, como en la creación de modelos y piezas que no estarán expuestas a altas temperaturas o esfuerzos mecánicos extremos [27].

4.1.2.3 Descripción de las impresoras 3D utilizadas

Se emplearon varias impresoras 3D durante el proceso de fabricación del hexápodo. La Creality CR-10S destaca por su gran volumen de construcción (300 x 300 x 400 mm) y su buena relación calidad-precio, lo que la hace ideal para imprimir piezas grandes. Además, cuenta con una estructura robusta de aluminio que garantiza la estabilidad durante el proceso de impresión [28] .

La BQ Witbox 2 es conocida por su excelente calidad de impresión y facilidad de calibración. Esta impresora tiene un volumen de construcción de 297 x 210 x 200 mm. Su estructura completamente cerrada y robusta proporciona una alta rigidez y estabilidad térmica durante la impresión [29].

La Prusa i3 es reconocida por su precisión y rapidez, siendo una de las impresoras más populares en el mercado de impresión 3D. Ofrece una excelente calidad de impresión y es compatible con una amplia gama de materiales, lo que la hace versátil para diversos proyectos [30].

Por último, la Hephestos, basada en el diseño de la Prusa i3, es una impresora de código abierto que se destaca por su fiabilidad y capacidad de personalización. Permite ajustes precisos y es ideal para proyectos que requieren una gran flexibilidad en la configuración de impresión [31].

Figura 61. Impresora Hephestos. Fuente: [31]

Figura 60. Impresora Prusa i3. Fuente: [30]

4.1.2.4 Problemas encontrados durante la impresión y soluciones

Durante el proceso de impresión, se encontraron varios problemas. Uno de los más comunes fue la falta de adhesión de las piezas a la cama de impresión, lo que provocaba que se despegaran durante la impresión. Para solucionar este problema, se utilizó una combinación de pegamento en barra y laca para el cabello en la cama de impresión, lo que mejoró significativamente la adhesión. Otro problema frecuente fue la obstrucción del extrusor. Esto se resolvió limpiando regularmente el extrusor y utilizando filamento de calidad y curarlos en un secador de filamentos para evitar impurezas que pudieran causar bloqueos.

4.1.2.5 Tiempo promedio de impresión para las diferentes piezas

A continuación, se muestra una tabla con el tiempo promedio de impresión para las diferentes piezas, suponiendo que todas las impresiones se realizan en la Prusa, una de las impresoras más rápidas utilizadas. El parámetro de tiempo se ha obtenido del software UltiMaker Cura:

Pieza	Tiempo estimado (Prusa i3)
Caja servo	1h 43m
U 68	1h 22m
U 71.4	1h 22m
Fémur (parte inferior)	2h 6m
Fémur (parte superior)	37m
Tibia	2h 51mm
Cuerpo (parte inferior)	4h 47m
Cuerpo (parte superior)	11h

Tabla 2. Tiempo estimado de impresión para las piezas que componen el robot hexápodo.

4.2 Hardware electrónico

En esta sección se detallarán los componentes electrónicos utilizados en el diseño y construcción del robot hexápodo, incluyendo el microcontrolador ESP32, los módulos PWM y motores, así como el sistema de alimentación, baterías y cableado.

4.2.1 Microcontrolador ESP32

El ESP32 es un microcontrolador avanzado que integra Wi-Fi y Bluetooth, ofreciendo una solución robusta para aplicaciones de robótica e IoT. En este proyecto, el ESP32 se utilizó en lugar de cualquier otro microcontrolador debido a sus numerosas ventajas, que lo hacen más adecuado para las necesidades del robot hexápodo.

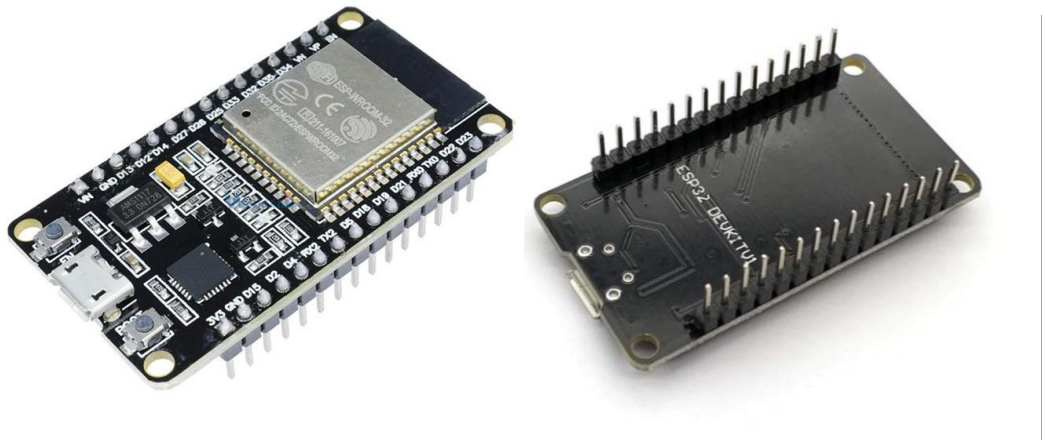


Figura 62. Microcontrolador ESP32. Fuente: [32]

4.2.1.1 Consideraciones iniciales

Inicialmente, se consideró y se puso en práctica el uso de un Arduino Mega junto con un shield que facilitara la conexión de los motores y evitara la necesidad de módulos generadores de señales PWM. Sin embargo, el shiel mostró ser poco fiable en la calidad de las conexiones y el Arduino Mega carecía de la potencia de cómputo requerida para el cálculo de trayectorias y movimiento fluido del hexápodo.

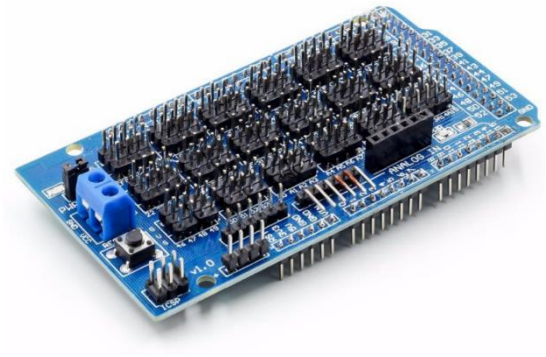


Figura 64. Shield de expansión para Arduino Mega (consideración inicial). Fuente: [33]

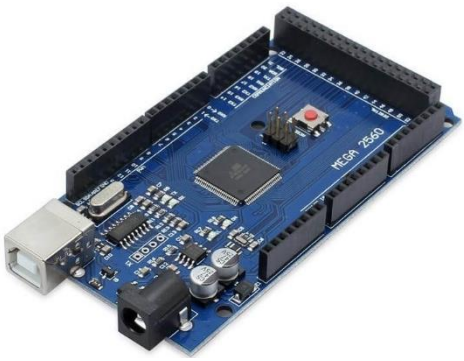


Figura 63. Microprocesador Arduino Mega (consideración inicial). Fuente: [34]

También se consideró y se implementó el uso de una Raspberry Pi 2 Model B para calcular la cinemática inversa y generar trayectorias, mientras el Arduino ejecutaba los movimientos recibidos mediante comunicación serial, de acuerdo con los cálculos realizados (se justifica más abajo la elección final en comparación con la Raspberry y el Arduino).

Sin embargo, se decidió generar archivos con la información de movimiento y guardarlos en la memoria de un ESP32, prescindiendo así de la Raspberry Pi 2 y sustituyendo al Arduino Mega por un microcontrolador mucho más rápido. Esta decisión simplificó el diseño, redujo la complejidad del sistema al eliminar la necesidad de comunicación serial entre dos dispositivos diferentes y redujo tiempos de ejecución, lo que se traduce en mayor fluidez del hexápodo. Además, que la Raspberry Pi 2 no dispone de Wi-Fi ni Bluetooth por lo que teníamos que usar el módulo Bluetooth HC-05.

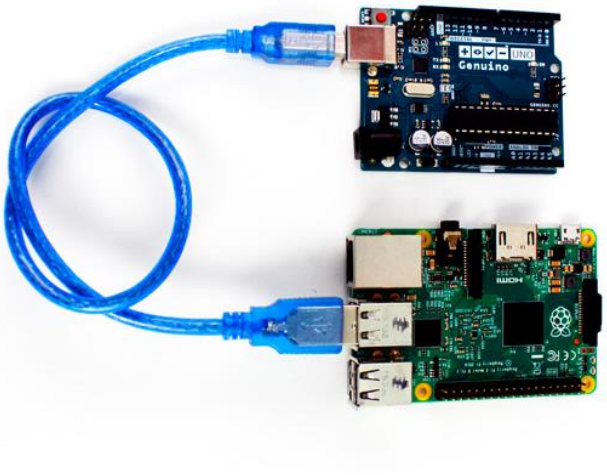


Figura 66. Arduino y Raspberry Pi conectados por comunicación serial (consideración inicial). Fuente: [35]

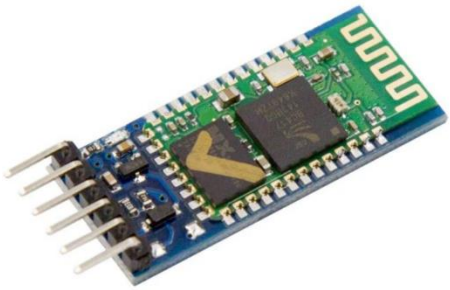


Figura 65. Módulo Bluetooth HC-05 (consideración inicial). Fuente: [36]

4.2.1.2 Ventajas del ESP32 frente al Arduino Mega

El ESP32 cuenta con un procesador dual-core Tensilica Xtensa LX6 que puede operar hasta a 240 MHz, lo cual es significativamente más rápido que el ATmega2560 del Arduino Mega, que opera a 16 MHz [37].

Esta diferencia en la velocidad de procesamiento permite al ESP32 manejar tareas más complejas y realizar múltiples operaciones simultáneamente con mayor eficiencia. La mayor velocidad de procesamiento permitió lecturas y cálculos más rápidos, esenciales para el control preciso de los movimientos del hexápodo.

Además, el ESP32 ofrece 520 KB de SRAM y típicamente 4 MB de flash, comparado con los 8 KB de SRAM y 256 KB de flash del Arduino Mega, proporcionando una mayor capacidad para almacenar y ejecutar programas más complejos [37].

La mayor memoria RAM permitió almacenar y gestionar archivos con parámetros de movimiento complejos y otros datos necesarios para la navegación del robot.

Como ya se mencionó anteriormente una de las mayores ventajas del ESP32 es su conectividad inalámbrica integrada. Incluye tanto Wi-Fi como Bluetooth, eliminando la necesidad de módulos adicionales para estas funciones. Esta capacidad integrada redujo la complejidad del diseño y también disminuyó los costos y el espacio requerido para el hardware electrónico.

El ESP32 ofrece 34 pines GPIO que pueden ser configurados para diversas funciones como ADC, DAC, I2C, UART, y SPI, proporcionando una mayor flexibilidad en comparación con los 54 pines digitales del Arduino Mega, de los cuales 15 son PWM [38]. La capacidad de tener múltiples entradas analógicas y digitales facilita la integración de varios sensores y actuadores, lo que es necesario para un robot hexápodo que requiere un control preciso de sus movimientos y sensores. En este caso se usaron los pines de destinados para la comunicación I2C los cuales fueron conectados a los módulos que generaban las señales PWM.

Característica	ESP32	Arduino Mega
Procesador	Dual-Core Tensilica Xtensa LX6 a 240 MHz	ATmega2560 a 16 MHz
Memoria RAM	520 KB	8 KB
Memoria Flash	4 MB	256 KB
Wi-Fi	Integrado	Necesita módulo externo
Bluetooth	Integrado	Necesita módulo externo
GPIOs	34 pines configurables	54 pines digitales
Lenguajes de Programación	C, C++, Python, Lua	C, C++
Modos de Ahorro de Energía	Múltiples modos de ahorro de energía	Limitado
Voltaje de Operación	3.3 V	5 V
Conectividad USB	USB-UART	USB Tipo B
Aplicaciones Comunes	IoT, dispositivos portátiles, hogares inteligentes	Proyectos educativos, prototipos avanzados

Tabla 3. Características del ESP32 frente al Arduino Mega.

Otra ventaja significativa del ESP32 es su flexibilidad en programación. Soporta múltiples lenguajes de programación como C, C++, Python, y Lua. Esta compatibilidad hizo que la transición desde el Arduino Mega fuera muy cómoda, ya que no fue necesario aprender un nuevo entorno de desarrollo ni un nuevo lenguaje de programación. De hecho, fue posible reutilizar muchos de los códigos fuente previamente elaborados, lo que aceleró significativamente el proceso de desarrollo y permitió centrarse en la optimización del hardware y el software del hexápodo.

4.2.2 Motores

Los motores son una parte fundamental de la estructura de cualquier robot, y una correcta elección de estos componentes es determinante para el desempeño eficiente y efectivo del sistema. En el caso del robot hexápodo, la selección de los motores adecuados fue esencial para asegurar que el robot pudiera moverse con precisión, velocidad y estabilidad. A continuación, se detallan las decisiones y características que llevaron a la selección final de los servomotores utilizados en este proyecto.

4.2.2.1 Decisión de utilizar servomotores

Para el proyecto del robot hexápodo, se consideraron varias opciones de motores antes de decidirse por servomotores. Inicialmente, se pensó en utilizar motores paso a paso 28BYJ-48 de 5V compatibles con Arduino debido a su precisión en el control de posición.

Figura 67. Motor paso a paso 28BYJ-48 (opción descartada para el proyecto). Fuente: [40]

Sin embargo, durante las pruebas se observó que, a pesar de poder configurar la velocidad de barrido de los ángulos al máximo, los motores paso a paso no eran lo suficientemente rápidos para las necesidades del proyecto y sus relaciones torque-peso no eran satisfactorias.

Para lograr un paso rápido y por poseer mejores relaciones torque-peso, se optó por usar servomotores, aunque esto implicara enfrentar mayores vibraciones y menos suavidad en el movimiento comparado con los motores paso a paso.

4.2.2.2 Primeros servomotores utilizados: MG995

Los primeros servomotores utilizados fueron los MG995, conocidos por su uso en aeromodelismo y aplicaciones robóticas de bajo costo. Sin embargo, estos servos no resultaron ser adecuados para el proyecto por dos razones principales.

Por un lado, debido a problemas de control, ya que los servos MG995 presentaban problemas de overshooting al recibir comandos de ángulos, lo que complicaba el control preciso del robot.

Y por otro lado debido a problemas relacionados con un torque insuficiente. Con un torque máximo de 11 kg/cm, los MG995 no proporcionaban la fuerza necesaria para mantener el robot estable y en movimiento correcto bajo la carga de su propio peso.

Figura 68. Izquierda: Servomotor MG995. Derecha: Servomotor MG995 junto a los attachment para el cabezal.(Opción descarta para el proyecto) Fuente: [41]

Las características técnicas del MG995 incluyen:

- Tipo: Analógico.
- Material de los ejes: Plástico.
- Carcasa: Plástico.
- Bearings: Single Ball Bearing.
- Dead Band: 5 µs.
- Voltaje de operación: 4.8V a 7.2V.
- Torque de parada: 9.4 kg/cm (4.8V); 11 kg/cm (6V).
- Velocidad de operación: 0.2 s/60º (4.8V), 0.16 s/60º (6V).
- Peso: 55 g.
- Dimensiones: 40.7 x 19.7 x 42.9 mm.
- Tipo de motor: Brushed.
- Waterproof: No.

4.2.2.3 Servomotores finales seleccionados

Dado que los MG995 no cumplían con los requisitos del proyecto, se optó por servomotores más potentes:

- DS-S020A-C de 180 grados y 30 kg/cm para Coxa y Tibia: Estos servomotores proporcionan un torque significativo de 30 kg/cm, suficiente para soportar las cargas en las articulaciones de la coxa y la tibia.

- SPT5435LV de 180 grados y 35kg/cm para Fémur: Elegidos por su mayor torque de 35 kg/cm, estos servos son ideales para las articulaciones del fémur, que soportan la mayor carga durante el movimiento del robot.

Figura 70. Servomotor DS-S020A-C. Fuente: [42]

Figura 69. Servomotor SPT5435LV. Fuente: [43]

Las ventajas de estos nuevos servomotores incluyen:

- Mayor Torque: Asegura que el robot pueda mantenerse estable y moverse eficientemente bajo carga.
- Mejor Control: Menos problemas de overshooting y mayor precisión en el control de ángulos.
- Durabilidad: Diseño robusto adecuado para aplicaciones exigentes en robótica.
- Tipo: Digital (mejor precisión y respuesta que los analógicos).
- Material de los ejes: Metal (mayor durabilidad y resistencia).
- Carcasa: Metal y plástico (para una combinación de durabilidad y peso ligero).
- Bearings: Dual Ball Bearing (mejora en la suavidad y durabilidad).
- Dead Band: 4 µs (mejor precisión en la posición).
- Voltaje de operación: 4.8V a 7.4V (rango de operación más amplio).
- Velocidad de operación: 0.16 s/60º (4.8V), 0.14 s/60º (6V).
- Peso: 60 g (DS-S020A-C), 65 g (SPT5435LV).
- Tipo de motor: Coreless (mayor eficiencia y respuesta).
- Waterproof: Sí (mejor protección en condiciones adversas).

Característica	MG995	DS-S020A-C	SPT5435LV
Torque Máximo	11 kg/cm	30 kg/cm	35 kg/cm
Tipo	Analógico	Digital	Digital
Material de los ejes	Plástico	Metal	Metal
Carcasa	Plástico	Metal y plástico	Metal y plástico
Bearings	Singles ball bearing	Dual ball bearing	Dual ball bearing
Dead Band	5 us	4 us	4 us
Voltaje de operación	4.8V a 7.2V	4.8V a 7.4V	4.8V a 7.4V
Velocidad de operación	0.2 s/60º (4.8V), 0.16 s/60º (6V)	0.15 s/60º (4.8V), 0.12 s/60º (6V)	0.16 s/60º (4.8V), 0.14 s/60º (6V)
Peso	55 g	60	65
Dimensiones	40.7 x 19.7 x 42.9 mm	40 x 20 x 38 mm	40.5 x 20 x 40,5 mm
Tipo de motor	Brushed	Coreless	Coreless
Waterproof	No	Sí	Sí

Tabla 4. Comparación de las características de los servomotores MG995 (descartado), DS-S020A-C (utilizado) y SPT5435LV (utilizado)

4.2.2.4 Calibración de los motores

La calibración de los servomotores es esencial para asegurar que el robot hexápodo se mueva con precisión y que cada servomotor apunte al ángulo correcto cuando se le envía una señal PWM específica. A continuación, se describe el proceso de calibración, los materiales empleados y los pasos seguidos para calibrar los servomotores.

Materiales Empleados:

- HJ Servo Tester
- Piezas impresas en 3D (aguja y base indicadora de ángulos)
- Servomotores

Figura 71. De izquierda derecha: Pieza impresa en 3D (base superior del servomotor) blanca, pieza impresa en 3D (aguja) negra y Servo Tester HJ. Fuente: Elaboración propia.

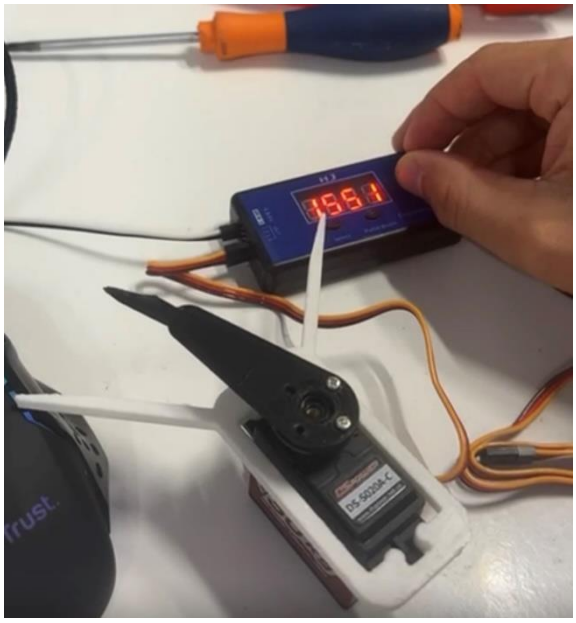


Figura 72. Proceso de calibración de los servomotores. Fuente: Elaboración propia.

Proceso de Calibración:

1. Configuración inicial:

Conectar el servomotor al HJ Servo Tester. Este dispositivo es conveniente porque genera señales PWM ajustables, necesarias para calibrar los servomotores.

2. Preparación de los indicadores de ángulo:

Utilizar una aguja impresa en 3D que se ensambla en la cabeza del servomotor. Esta aguja servirá para indicar el ángulo al que apunta el servomotor.

Colocar otra pieza impresa en 3D en la base superior del servomotor. Esta pieza tiene marcas que indican los ángulos de 45º y 135º.

3. Generación de señales PWM:

Usar el HJ Servo Tester para generar una señal PWM con un ciclo de trabajo específico, por ejemplo, 1500 us, y ensamblar la aguja en esa posición. Idealmente, esta señal debería mover el servomotor a 90º, pero los servomotores pueden variar, apuntando a diferentes ángulos.

4. Ajuste Fino de los Ángulos:

Ajustar la señal PWM con el HJ Servo Tester para que la aguja del servomotor apunte exactamente a 45º y 135º, utilizando las marcas de la pieza impresa en 3D.

Anotar los valores de la señal PWM (en microsegundos, μs) para los ángulos de 45º y 135º para cada servomotor.

5. Interpolación Lineal:

Con los valores anotados para los ángulos de 45º y 135º, realizar una interpolación lineal para calcular la señal PWM necesaria para cualquier ángulo entre 0º y 180º. La interpolación lineal permite establecer una relación precisa entre la señal PWM y el ángulo del servomotor, asegurando que cada servomotor apunte correctamente en cualquier ángulo dentro de su rango operativo.

6. Posicionamiento Final:

Ajustar el servomotor para que apunte a 90º utilizando la señal PWM correspondiente determinada durante la calibración. Esto deja el servomotor listo para el ensamblaje final en el robot hexápodo. Finalmente se etiqueta con su número correspondiente entre 1 y 18.

Resultados de la Calibración:

La calibración de los servomotores mediante este método asegura que cada motor responda con precisión a las señales PWM enviadas. Los valores obtenidos y la interpolación lineal permiten que el control del robot sea más exacto, mejorando su rendimiento y capacidad de movimiento.

Este proceso de calibración es fundamental para que el robot hexápodo mantenga la estabilidad y realice movimientos precisos. Si hallamos la fórmula general de la recta usando la interpolación lineal tenemos que:

INTERPOLACIÓN LINEAL	
45º	t_1
x	y
135º	t_2

Vemos que para un ángulo de 45º, el servo motor necesita una señal PWM con un ciclo de trabajo de $t_1\ \mu s$ y para un ángulo de 135º el servo motor necesita una señal PWM con un ciclo de trabajo de $t_2\ \mu s$. Por lo que para barrer xº hará falta una señal de $y\ \mu s$.

Por lo tanto, tenemos que:

$$\frac{45 - x}{45 - 135} = \frac{t_1 - y}{t_1 - t_2}$$

$$\frac{45 - x}{-90} = \frac{t_1 - y}{t_1 - t_2}$$

$$(45 - x) \cdot (t_1 - t_2) = -90(t_1 - y)$$

$$45t_1 - 45t_2 - xt_1 + xt_2 = -90t_1 + 90y$$

$$-90y = -90t_1 - 45t_1 + 45t_2 + xt_1 - xt_2$$

$$-90y = -135t_1 + 45t_2 + xt_1 - xt_2$$

$$y = \frac{3}{2}t_1 - \frac{1}{2}t_2 + \frac{-xt_1 + xt_2}{90}$$

$$y = x\frac{(t_2 - t_1)}{90} + \frac{3}{2}t_1 - \frac{t_2}{2}$$

Donde y representa el tiempo en μs del ciclo de trabajo de la señal PWM correspondiente para que el servomotor apunte al ángulo x. Las variables t_1 y t_2 representan los valores de tiempo en μs del ciclo de trabajo necesario para que el servomotor apunte hacia los 45 y 135 grados respectivamente, previamente obtenidos en la calibración.

Organizando en una tabla los valores obtenidos en la calibración para cada uno de los servomotores y sus rectas asociadas se tiene que:

Servo	$t_1(\mu s)$	$t_2(\mu s)$	Ecuación de la recta
1	923	1906	$y = \frac{983}{90}x + \frac{863}{2}$
2	963	1952	$y = \frac{989}{90}x + \frac{937}{2}$
3	881	1960	$y = \frac{1079}{90}x + \frac{683}{2}$
4	1047	2072	$y = \frac{205}{18}x + \frac{1069}{2}$
5	1063	2002	$y = \frac{313}{30}x + \frac{1187}{2}$
6	1000	1995	$y = \frac{199}{18}x + \frac{1005}{2}$
7	1010	2002	$y = \frac{496}{45}x + 514$
8	994	1953	$y = \frac{959}{90}x + \frac{1029}{2}$
9	998	1980	$y = \frac{491}{45}x + 507$
10	966	1970	$y = \frac{502}{45}x + 464$
11	945	1934	$y = \frac{989}{90}x + \frac{901}{2}$
12	958	1938	$y = \frac{98}{9}x + 468$
13	872	1955	$y = \frac{361}{30}x + \frac{661}{2}$
14	1035	2005	$y = \frac{97}{9}x + 550$
15	1047	2095	$y = \frac{524}{45}x + 523$
16	1046	2067	$y = \frac{1021}{90}x + \frac{1071}{2}$
17	997	2011	$y = \frac{169}{15}x + 490$
18	935	1927	$y = \frac{496}{1}x + 490$

Tabla 5. Ecuaciones de las rectas, resultado de la calibración para cada servomotor.

Si graficamos las rectas tenemos que:

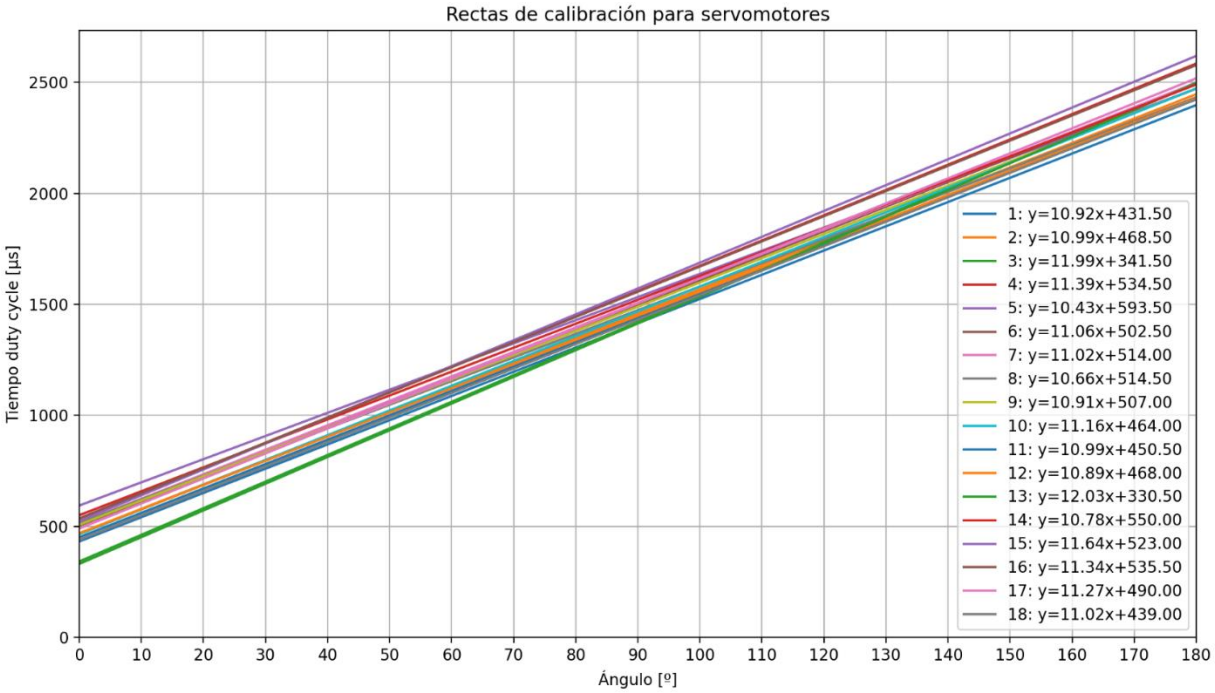


Figura 73. Rectas de calibración de cada servomotor. Fuente: Elaboración propia.

La gráfica de calibración para los servomotores muestra las rectas obtenidas para cada motor. Cada línea en la gráfica representa la relación entre el ángulo y el tiempo de duty cycle (en microsegundos) para un servomotor específico.

Las rectas de calibración muestran que hay variabilidad en cómo cada servomotor responde a las señales PWM.

4.2.3 Módulo PWM

Para controlar los servomotores del robot hexápodo, es esencial generar señales PWM (Pulse Width Modulation). Aunque es posible usar la librería <Servo.h> del entorno de desarrollo Arduino para generar estas señales directamente desde el microcontrolador, resulta mucho más conveniente utilizar módulos PWM dedicados, como el PCA9685.

Estos módulos simplifican el cableado y la programación y proporcionan un control más preciso y estable de los servomotores.

4.2.3.1 PCA9685

El PCA9685 es un controlador PWM de 16 canales desarrollado por NXP Semiconductors y popularizado en la comunidad de makers por Adafruit. Este módulo permite controlar hasta 16 salidas PWM de manera independiente, lo que lo hace ideal para aplicaciones que requieren el manejo de múltiples servomotores, como es el caso del robot hexápodo [44].

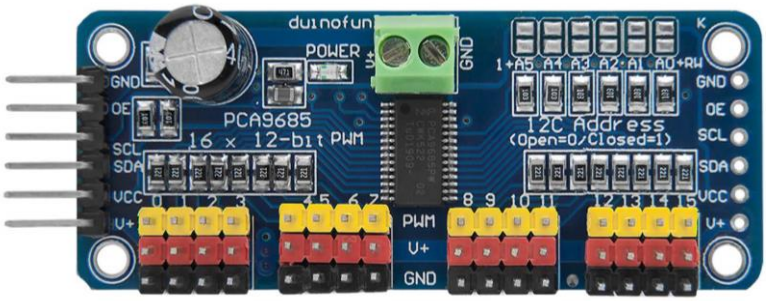


Figura 74. Módulo generador de señales PWM PCA9685. Fuente: [45]

El PCA9685 es un controlador PWM con una serie de ventajas notables que lo hacen una opción popular para proyectos de control de movimiento y robótica. Una de sus principales características es la capacidad de manejar 16 canales independientes, lo que permite generar señales PWM con ciclos de trabajo individualizados para cada canal. Esto significa que se pueden controlar simultáneamente hasta 16 servomotores, lo que es ideal para aplicaciones complejas que requieren precisión y sincronización.

Además, el PCA9685 utiliza una interfaz de comunicación I2C, lo que reduce significativamente la cantidad de pines necesarios para la conexión con el microcontrolador. Esta característica nos facilita la expansión del sistema y permite una integración más sencilla del hardware electrónico.

Otra ventaja importante es la frecuencia configurable de la señal PWM. A través del código, se puede ajustar la frecuencia, lo cual es adecuado para la mayoría de las aplicaciones de control de movimiento [45]. Para el proyecto se usó una frecuencia de 50Hz.

El PCA9685 también destaca por su voltaje de operación flexible, siendo compatible tanto con sistemas de 3.3V como de 5V. Esta versatilidad permite su uso con una amplia gama de microcontroladores, aumentando su aplicabilidad en diversos proyectos.

Finalmente, el diseño del PCA9685 se enfoca en el bajo consumo de energía, una característica importante para aplicaciones móviles y autónomas, como los robots hexápodos. Este bajo consumo es básico para maximizar la eficiencia y la duración de la batería en dispositivos que requieren movilidad [44].

4.2.3.1.1 Funcionamiento

El PCA9685 tiene 16 canales PWM que se pueden utilizar para controlar servomotores u otros dispositivos que requieran señales PWM. En el proyecto del hexápodo, se utilizaron dos módulos PCA9685, ya que cada uno ofrece 16 canales y se necesitan 18 servos para controlar todas las patas del robot.

Conexiones básicas:

- VCC: Conectar a 3.3V o 5V del microcontrolador (según la lógica del sistema).
- GND: Conectar al ground del microcontrolador.
- SCL: Conectar al pin SCL del microcontrolador.
- SDA: Conectar al pin SDA del microcontrolador.

Configuración de la dirección I2C:

Cada PCA9685 tiene una dirección I2C predeterminada de 0x40. Al usar dos módulos, se tuvo que cambiar la dirección de uno de ellos a 0x41. Esto se logró soldando un pin en el módulo para cambiar su dirección concretamente el A0.

4.2.3.1.2 Envío de Comandos de Control

El PCA9685 tiene una resolución de 12 bits, lo que implica que el ciclo completo de la señal PWM se divide en 4096 cuentas, numeradas de 0 a 4095.

Se utilizó la librería “”Adafruit_PWMServoDriver” para controlar el módulo. Esta librería permite configurar la frecuencia y los ciclos de trabajo de cada canal PWM de manera sencilla.

Para enviar comandos de control a los servomotores, se utiliza la función “setPWM()”.

Esta función recibe tres parámetros:

- Canal: El canal PWM que se desea controlar (0-15 para cada PCA9685).
- On: El tiempo (en ticks) cuando la señal PWM debe encenderse. Este parámetro se establece en 0 para simplificar el control en nuestro caso.
- Off: El tiempo (en ticks) cuando la señal PWM debe apagarse, determinando así la duración del pulso.

El uso del parámetro “on” como 0 simplifica el cálculo de las señales PWM necesarias para cada ángulo de los servomotores.

El PCA9685 es capaz de generar señales PWM a una frecuencia específica. En este caso, el módulo está configurado para operar a 50 Hz.

Esto significa que cada ciclo de la señal PWM tiene una duración de 1/50 segundos, lo cual equivale a 0,02 segundos, o 20 milisegundos. En términos de microsegundos, esto corresponde a 20.000 microsegundos por ciclo.

Utilizando las rectas de calibración de los servomotores, se calcula el valor en microsegundos (us) para el ángulo deseado. Ahora, para convertir un valor en microsegundos a cuentas, primero debemos determinar cuántos microsegundos corresponden a una sola cuenta.

Sabemos que 20,000 microsegundos equivalen a 4096 cuentas. Por lo tanto, podemos calcular los microsegundos por cuenta dividiendo el total de microsegundos entre el número de cuentas:

$$\frac{20000\ \mu s}{4096\ cuentas} = 4{,}88\ \mu s/cuentas$$

Con este valor, podemos convertir cualquier cantidad de microsegundos a cuentas dividiendo el número de microsegundos entre los microsegundos por cuenta.

Este proceso permite traducir valores de tiempo en microsegundos a un formato comprensible y manejable por el módulo PCA9685.

4.2.3.1.3 Corriente limitada en el PCA9685

Durante las pruebas, se observó que los módulos PCA9685 tenían un MOSFET que actuaba como circuito de protección en la entrada de alimentación.

Este MOSFET limitaba la corriente que llegaba a los servomotores, generando problemas como movimientos erráticos, movimientos a destiempo o incluso falta de movimiento. Para resolver este problema, se realizó un puente en la placa con un cable, soldando la entrada de alimentación de los servos directamente a la alimentación de los servomotores, evitando el paso por el MOSFET.

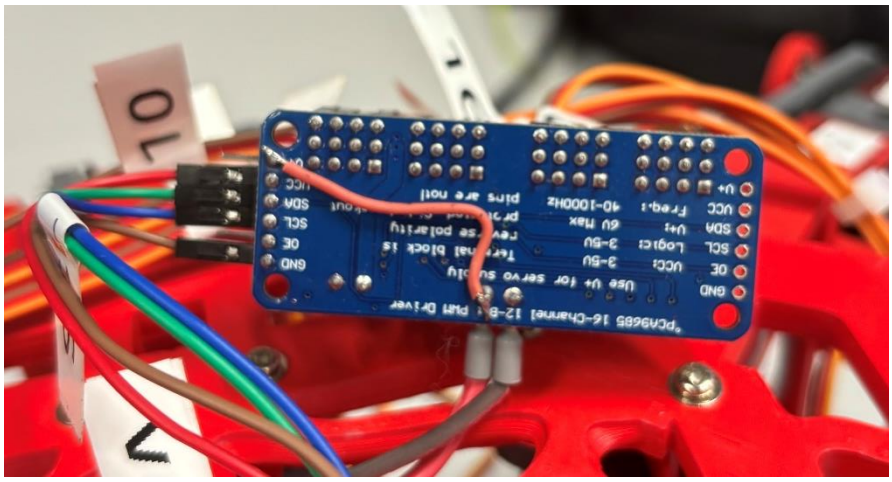


Figura 75. Puente en el módulo PCA9685. Fuente: Elaboración propia.

4.2.4 Alimentación y baterías

La elección y gestión de la fuente de alimentación son aspectos críticos para el funcionamiento eficiente y seguro del robot hexápodo. A continuación se describen los componentes principales utilizados para la alimentación del robot, incluyendo las baterías, conectores y reguladores de tensión.

4.2.4.1 Baterías Samsung INR21700-40T

Para alimentar el robot hexápodo, se utilizaron dos baterías SAMSUNG INR21700-40T SDI conectadas en serie. Estas baterías de litio tienen una alta capacidad y un excelente rendimiento en aplicaciones de alta corriente, lo que las hace ideales para proyectos robóticos.

Características de la Batería SAMSUNG INR21700-40T SDI:

- Capacidad Nominal: 4000mAh.
- Voltaje Nominal: 3.6V.
- Corriente de Descarga Continua: 35A.
- Dimensiones: 21mm de diámetro y 70mm de longitud.

- Peso: Aproximadamente 68g.

Estas características proporcionan una buena combinación de capacidad y capacidad de descarga, lo que asegura un funcionamiento prolongado y confiable del robot bajo cargas elevadas. Al conectar dos de estas baterías en serie, se obtiene una tensión de 7.2V, suficiente para alimentar los componentes del robot.

Figura 77. Batería Samsung INR21700-40T.
Fuente: [46]

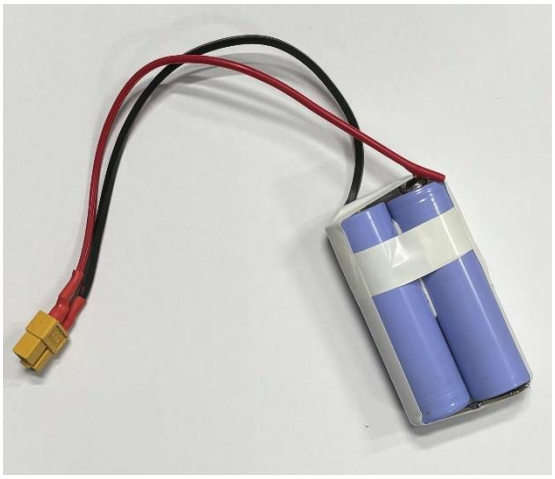


Figura 76. Baterías Samsung INR21700-40T en serie.
Fuente: Elaboración propia

4.2.4.2 Conectores XT-60

Se usaron conectores XT-60 para las baterías los cuales son ampliamente utilizados en aplicaciones que requieren una conexión robusta y confiable para el paso de corriente elevada. Una de las principales ventajas de estos conectores es su capacidad para soportar hasta 60 amperios de corriente continua, lo que los hace ideales para proyectos que demandan alta potencia, como es el caso de un robot hexápodo.

La baja resistencia de contacto de los conectores XT-60 asegura que haya mínima pérdida de energía durante la transmisión de corriente, lo cual es crucial para mantener la eficiencia del sistema.

Además, los conectores XT-60 están fabricados con materiales resistentes al calor y a la corrosión, lo que incrementa su durabilidad y fiabilidad a lo largo del tiempo. Esta característica es particularmente beneficiosa en entornos donde las condiciones pueden variar, como en proyectos de robótica que implican movimientos constantes y posibles exposiciones a diferentes temperaturas.

La facilidad de uso es otra ventaja significativa; los conectores XT-60 se ensamblan y desensamblan con facilidad, proporcionando una conexión segura y evitando desconexiones accidentales durante el funcionamiento del robot [47].

Figura 79. Par de conectores XT-60. Fuente: [48]

Figura 78. Par de cables con conectores XT-60. Fuente: [49]

4.2.4.3 Carga de las baterías

Para la carga de las baterías, se utilizó el cargador Sigma AC/DC Charger Pro-Peak.

El Sigma AC/DC Charger Pro-Peak es un cargador versátil y eficiente diseñado para una amplia gama de baterías, incluyendo Li-Po, Li-Ion, Ni-Cd, Ni-MH y baterías de plomo-ácido. Este dispositivo permite cargar las baterías de manera segura, lo cual es primario para mantener el rendimiento y la longevidad de las baterías utilizadas en el robot hexápodo.

Una de las características destacadas del Sigma AC/DC Charger Pro-Peak es su compatibilidad con diferentes tipos de baterías y su capacidad para manejar múltiples celdas, proporcionando una flexibilidad considerable en su uso. El cargador puede operar con una entrada de 100-240V AC o 11.5-14.5V DC, lo que lo hace adaptable a diferentes fuentes de alimentación.

El control de carga del Sigma AC/DC Charger Pro-Peak es otro de sus puntos fuertes. Permite seleccionar el tipo de batería y ajustar la corriente de carga, asegurando que las baterías se carguen de manera óptima sin riesgo de sobrecarga. Además, cuenta con una pantalla LCD que muestra el estado de la carga y los parámetros de la batería en tiempo real, proporcionando información clara y precisa al usuario durante el proceso de carga.

Figura 80. Cargador Sigma AC/DC Charger Pro-Peak. Fuente: [67]

Figura 81. Cargando baterías en el laboratorio. Fuente: Elaboración propia.

4.2.4.4 Regulador de tensión Pololu U1V11A

El regulador de tensión Pololu U1V11A es un componente clave para proporcionar una alimentación estable y eficiente al microcontrolador ESP32 en el robot hexápodo. Este regulador es capaz de reducir la tensión de entrada, que en este caso proviene de las baterías de 7.2V, a una salida ajustable de 5V necesaria para el funcionamiento del ESP32.

Entre las características del Pololu U1V11A, destaca su amplio rango de entrada que va desde 2.5V hasta 12V, lo que permite su uso en diversas aplicaciones y con diferentes fuentes de alimentación [50].

La salida ajustable del regulador, que puede ser configurada entre 2.5V y 7.5V, ofrece flexibilidad en su implementación para diferentes requerimientos de voltaje.

Figura 82. Pololu Regulador de voltaje U1V11A. Fuente: [50]

El Pololu U1V11A también se distingue por su alta eficiencia, que minimiza la disipación de calor y maximiza la eficiencia energética del sistema. Esta eficiencia es clave en aplicaciones robóticas donde el manejo del calor y el consumo de energía deben ser optimizados para asegurar un rendimiento óptimo y una mayor duración de la batería.

Además, el regulador puede suministrar hasta 1.5A de corriente de salida, lo que es más que suficiente para alimentar el ESP32 y otros componentes de baja potencia que puedan estar conectados al mismo.

La incorporación de un conector micro USB al regulador facilita la conexión directa al ESP32, proporcionando una solución de alimentación limpia y organizada para el sistema robótico.

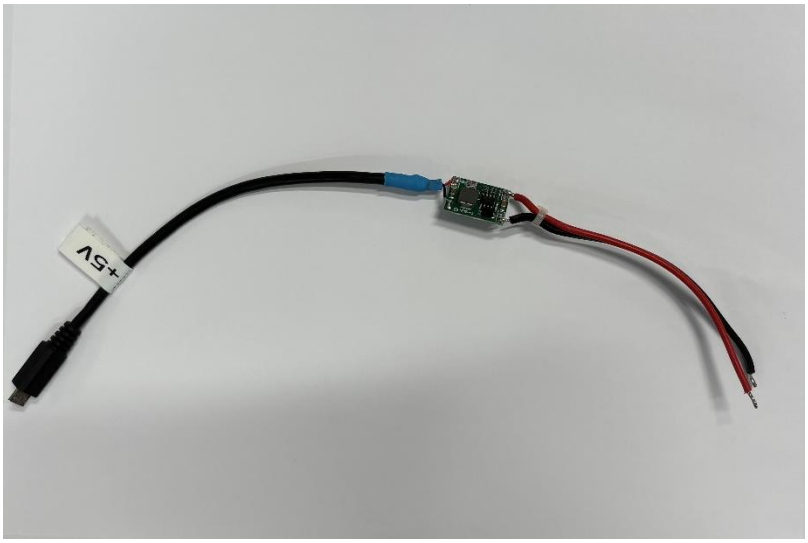


Figura 83. Conexión del regulador Pololu U1V11A con un conector micro USB. Fuente: Elaboración propia.

Estas soluciones de alimentación aseguran que todos los componentes del robot hexápodo funcionen de manera eficiente y estable, permitiendo un control preciso y una operación confiable.

4.2.5 Cableado

El cableado es una parte esencial en el ensamblaje y funcionamiento del robot hexápodo, ya que asegura la correcta interconexión y alimentación de todos los componentes electrónicos. En este apartado se describe el proceso detallado de cableado, desde la conexión de los servomotores hasta la integración del microcontrolador ESP32 con los módulos PWM y el sistema de alimentación.

4.2.5.1 Conexiones iniciales y problemas encontrados

Inicialmente, se utilizaron cables DuPont para realizar todas las conexiones del hardware electrónico del robot. Estos cables se conectaban directamente desde los servomotores a los módulos PWM y al microcontrolador ESP32. Sin embargo, durante las pruebas prácticas, se observó que estas conexiones eran débiles, generaban malos contactos y se soltaban fácilmente, lo que provocaba movimientos indeseados en las patas del robot debido a la inestabilidad de la conexión.

Para resolver los problemas de conexión, se decidió soldar todas las conexiones, garantizando un cableado más robusto y seguro. Para este propósito, se utilizaron punteras, pines, regletas electrónicas y termo retráctiles. Además, se emplearon bridas y mangueras espiraladas para ordenar y proteger los cables.

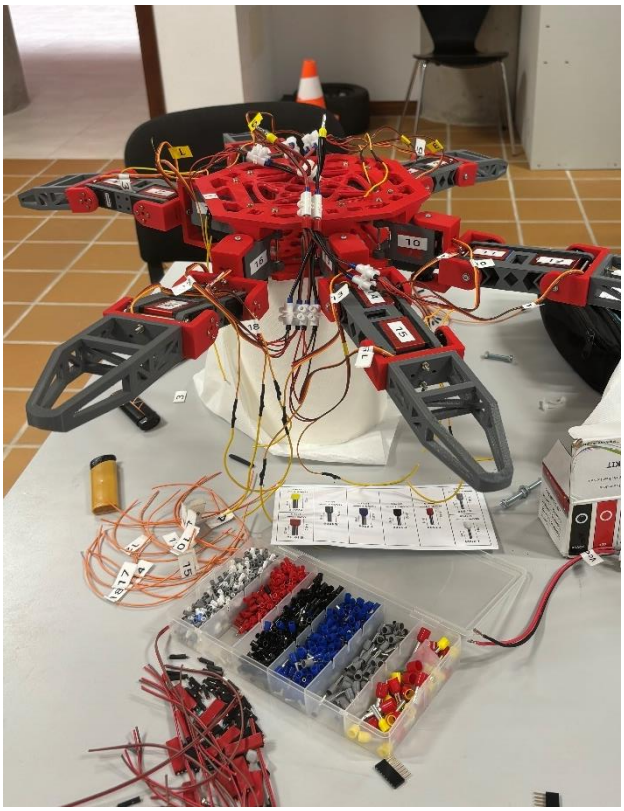


Figura 84. Proceso de cableado en el laboratorio I. Fuente: Elaboración propia.

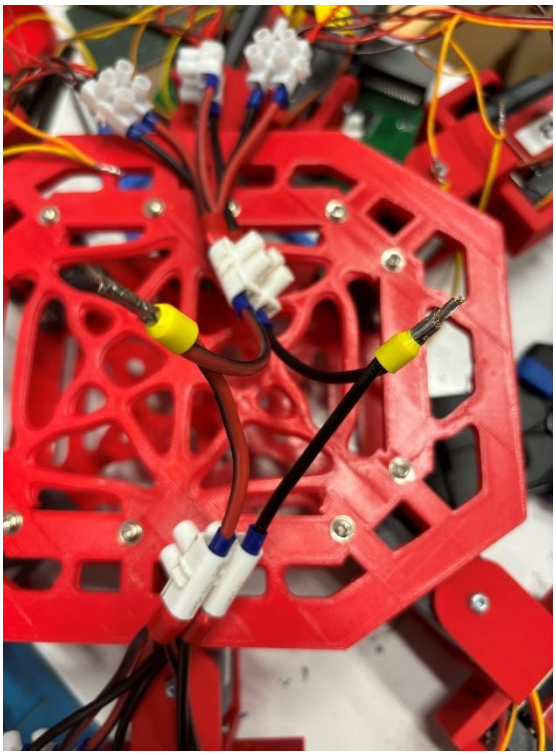


Figura 85. Punteras, clemas y cables utilizados. Fuente: Elaboración propia.

4.2.5.2 Interconexión de los Módulos PWM

Los módulos PWM, utilizados para generar las señales necesarias para el control de los servomotores, fueron soldados entre sí y unidos mediante pines.

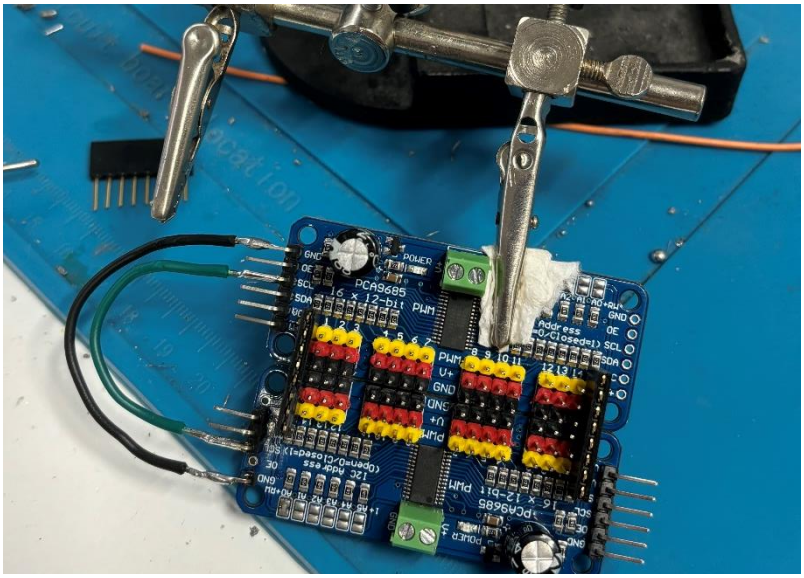


Figura 86. Proceso de interconexión y soldadura de los módulos PWM. Fuente: Elaboración propia.

4.2.5.3 Alimentación de los Servomotores

Inicialmente, la alimentación de los servomotores se realizaba a través de los módulos PWM, conectando los cables de VCC y GND de los servomotores a los módulos utilizando cables DuPont. Esta configuración también resultó ineficaz debido a la mala conexión de estos y a la reducida sección de los cables.

Para solucionar este problema, se optó por conectar directamente la alimentación de los servomotores a las baterías. Se organizó el cableado de tal manera que los cables de VCC y GND de cada pata se agrupasen en un solo punto. Posteriormente, se realizó el mismo procedimiento con tres patas consecutivas, agrupando las conexiones de ambos lados del hexápodo.

Finalmente, se obtuvieron dos cables principales (VCC y GND) capaces de alimentar a los 18 servomotores del robot desde las baterías. Durante este proceso, se utilizaron cables de mayor sección a medida que se acercaban a la batería: cables de 0,5 mm², 0,75 mm² y, finalmente, de 1,75 mm² para asegurar una adecuada capacidad de corriente.

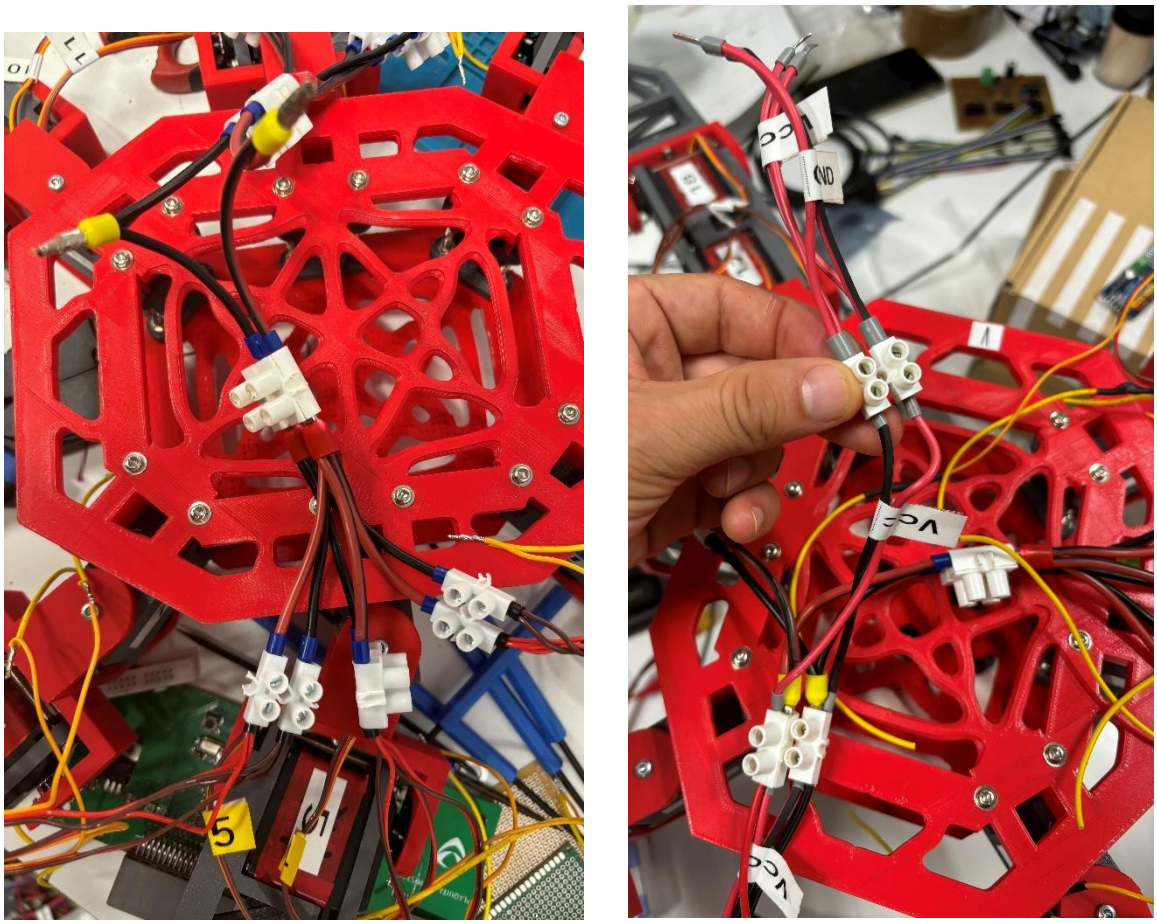


Figura 87. Tanto izquierda como derecha: Cableado de las patas y alimentación de los servomotores. Fuente: Elaboración propia.

4.2.5.4 Conexión de Señales PWM

Los cables de señal (comandos PWM) de los servomotores fueron alargados mediante soldaduras para alcanzar los módulos PWM situados en el centro del robot. Se utilizaron cables 20 AWG para estas conexiones, garantizando una buena transmisión de señal. Cada cable de señal se soldó a pines conectados a los módulos PWM, utilizando termo retráctiles para aislar y proteger las uniones.

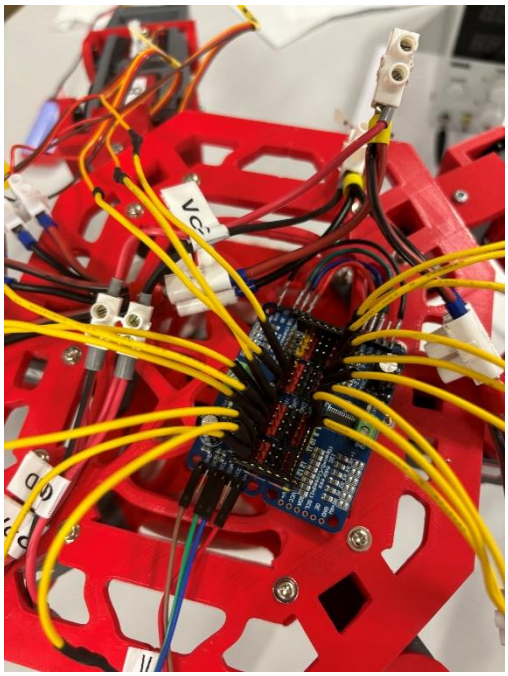


Figura 89. Conexión de cables de señal PMW al módulo PCA9685. Fuente: Elaboración propia.

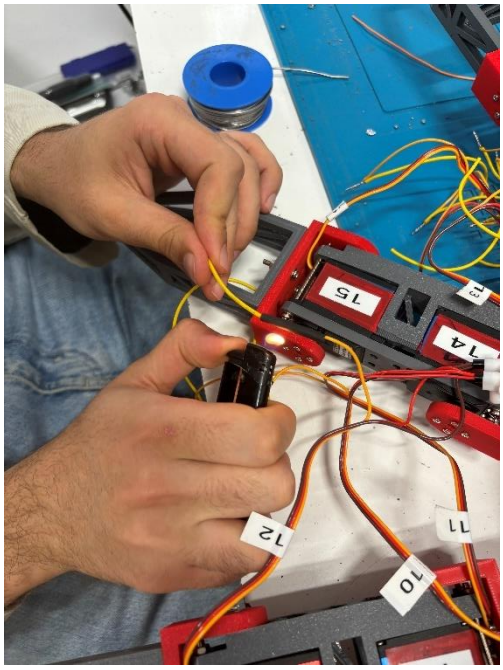


Figura 88. Proceso del cableado en el laboratorio II. Fuente: Elaboración propia.

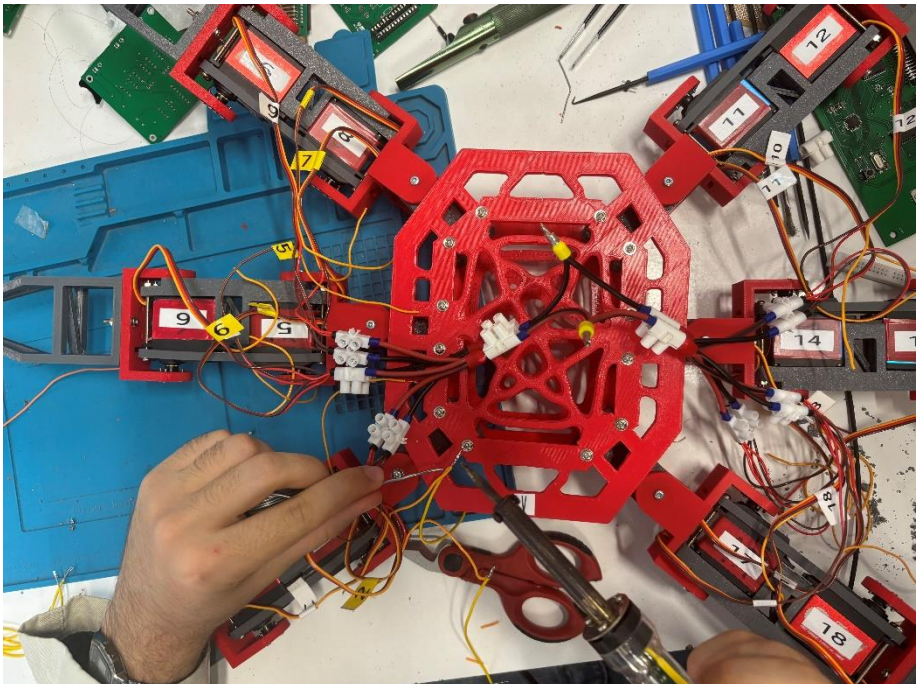


Figura 90. Proceso del cableado en el laboratorio III. Fuente: Elaboración propia.

4.2.5.5 Alimentación del ESP32

Finalmente, se conectó la tensión de la batería a la entrada del regulador que alimenta al ESP32. Las conexiones se realizaron utilizando el mismo enfoque de robustez: punteras y clemas para asegurar la fiabilidad de las conexiones eléctricas.

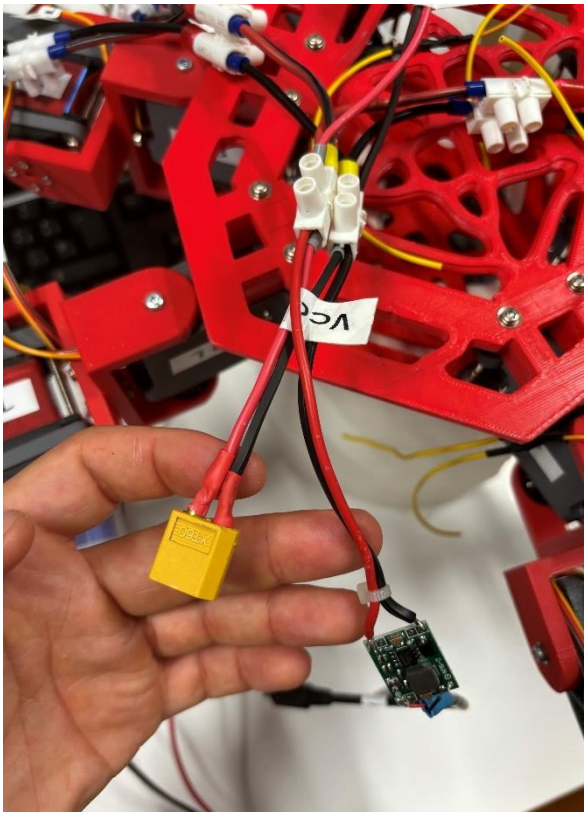


Figura 92. Conexionado del regulador a la tensión de la batería. Fuente: Elaboración propia.

Figura 91. Cableado terminado y vista del robot desde arriba. Fuente: Elaboración propia.

En el Anexo 10.5 Esquema de conexiones de muestra el esquema del cableado del robot hexápodo realizado en KiCad.

5 Software y algoritmos matemáticos

En este capítulo se detallarán los aspectos relacionados con el software desarrollado para el control y operación del robot hexápodo. Se explicarán las matemáticas y la lógica detrás del cálculo de la cinemática inversa y la generación de trayectorias, así como la implementación de scripts en Python y el desarrollo de la aplicación móvil en App Inventor.

5.1 Cálculo cinemático y trayectorias

Antes de adentrarnos en la cinemática inversa, es importante entender la cinemática directa. Aunque la cinemática directa no nos resuelve el problema de qué variables articulares asignar a los motores para alcanzar un punto específico, fue esencial en las primeras etapas del desarrollo ya que nos permitió dibujar gráficamente una pata del hexápodo y, posteriormente, al calcular las ecuaciones de la cinemática inversa, verificando que los resultados obtenidos coincidían con la posición esperada.

De esta manera, aseguramos que los ángulos calculados en la cinemática inversa para un punto en el espacio corresponden efectivamente a ese punto cuando se utilizan en la cinemática directa. Estas verificaciones e implementaciones se realizaron inicialmente usando el programa Octave antes de transferir las ecuaciones a scripts en Python para la generación de trayectorias.

5.1.1 Cinemática directa de una pata del hexápodo

La cinemática directa de una pata del hexápodo implica calcular la posición y orientación del extremo de la pata (efector final) en función de los ángulos de las articulaciones.

Se realizó un estudio de la cinemática directa utilizando el algoritmo de Denavit-Hartenberg (D-H).

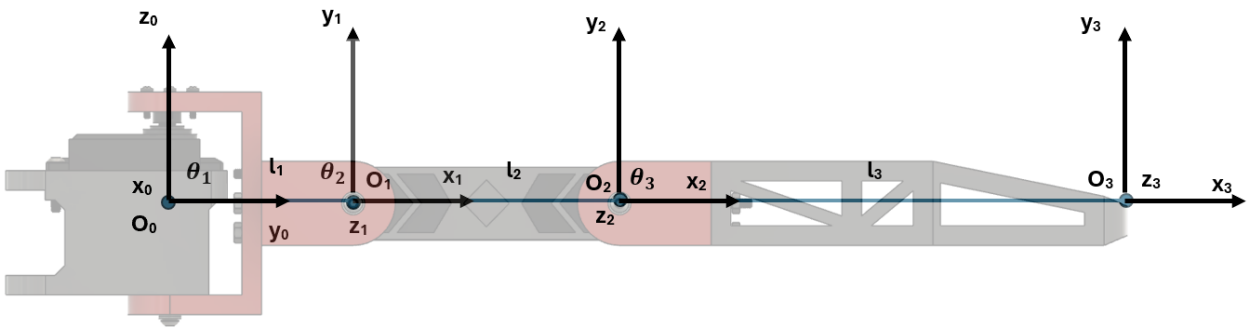


Figura 93. Representación de los distintos sistemas de coordenadas en la pata. Fuente: Elaboración propia.

Se define una función para calcular la matriz de transformación homogénea T_i de cada eslabón en función de los parámetros D-H:

$$T_i = \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Donde C se trata de la función $\cos(\square)$ y S se trata de la función $\sin(\square)$. Hallando los parámetros de D-H para esta configuración de tiene:

	1	2	3
d	0	0	0
θ	θ_1	θ_2	θ_3
a	l_1	l_2	l_3
α	$\pi/2$	0	0

Tabla 6. Parámetros de DH para las patas del robot hexápodo.

El valor de las distancias l_1, l_2 y l_3 se sacaron midiendo del diseño 3D dando como resultado 55,975 mm, 81,311 mm y 155,185 mm respectivamente.

Por lo tanto, para cada eslabón del manipulador, se calculan las matrices de transformación utilizando estos parámetros:

$$T_{01} = T(d_1, \theta_1, a_1, \alpha_1)$$

$$T_{12} = T(d_2, \theta_2, a_2, \alpha_2)$$

$$T_{23} = T(d_3, \theta_3, a_3, \alpha_3)$$

Luego se calculan las matrices de transformación acumuladas:

$$T_{02} = T_{01} \cdot T_{12}$$
$$T_{03} = T_{02} \cdot T_{23}$$

Posteriormente utilizando las matrices de transformación, se determina la posición del efector final:

$$O_{00} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$O_{11} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$O_{22} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$O_{33} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$O_{10} = T_{01} \cdot O_{11}$$
$$O_{20} = T_{02} \cdot O_{22}$$
$$O_{30} = T_{03} \cdot O_{33}$$

Donde O_{11}, O_{22} y O_{33} son los vectores de posición de los orígenes de los mundos 1, 2 y 3 respectivamente en el sistema de coordenadas homogéneo.

Las coordenadas del extremo de la pata son:

$$x_3 = O_{30}(1)$$
$$y_3 = O_{30}(2)$$
$$z_3 = O_{30}(3)$$

Donde el (1) en $O_{30}(1)$ indica que estamos tomando el primer elemento del vector O_{30}, que corresponde a la coordenada x del efector final en el espacio tridimensional. Similarmente, $O_{30}(2)$ y $O_{30}(3)$ corresponden a las coordenadas y y z , respectivamente.

Para verificar visualmente la cinemática directa, se dibujó la configuración de la pata en un gráfico 3D, conectando los puntos calculados para los orígenes de los mundos.

Se utilizó la función *plot3* para representar los eslabones y las articulaciones en un espacio tridimensional, asegurando que los resultados obtenidos sean coherentes con la configuración física de la pata del hexápodo.

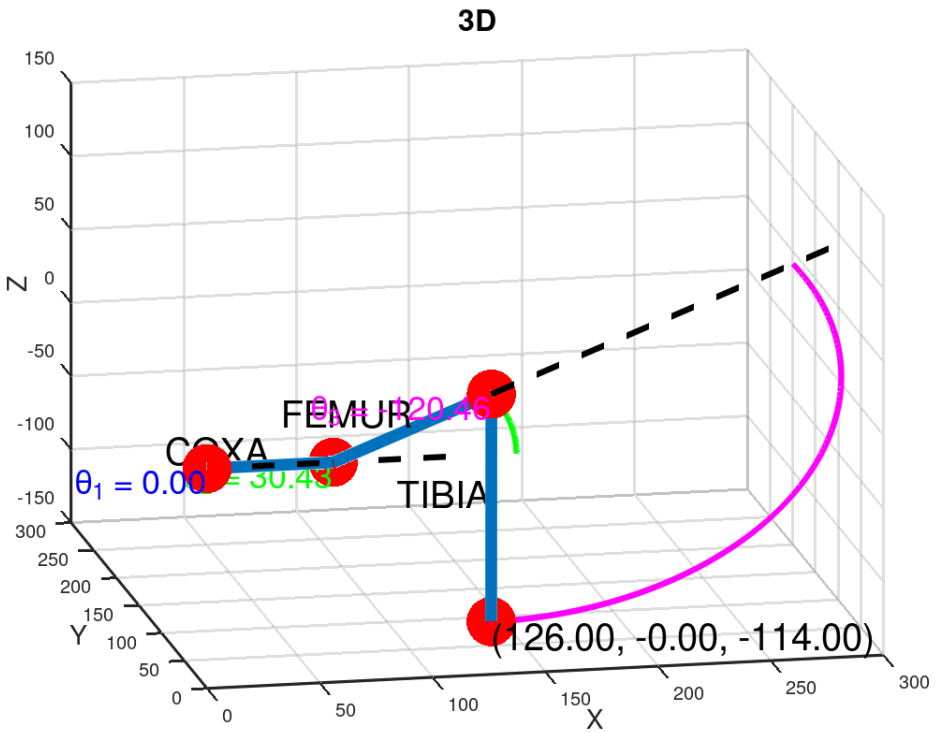


Figura 94. Representación gráfica de la cinemática inversa mediante Octave para unos valores de variables articuladas theta1:0º, theta2: 30º y theta3: -120º (Posición de reposo de la pata). Fuente: Elaboración propia.

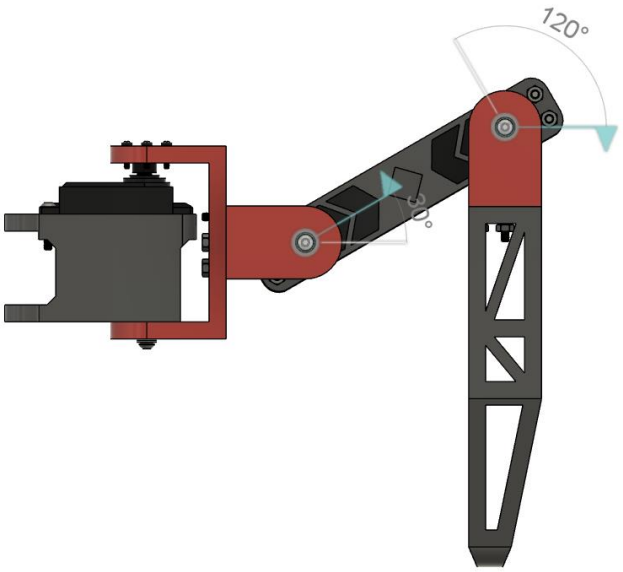


Figura 95. Representación de la pata en Fusion 360 en correspondencia con los valores de variables articuladas descritas en la figura superior. Fuente: Elaboración propia.

5.1.2 Cinemática inversa de una pata del hexápodo

El cálculo de la cinemática inversa es fundamental para el movimiento coordinado de las patas del hexápodo. En este apartado, se explicará cómo se derivan las ecuaciones matemáticas que permiten determinar los ángulos necesarios en cada articulación de las patas para alcanzar una posición deseada.

La cinemática inversa se utiliza para calcular los ángulos de las articulaciones a partir de una posición objetivo del extremo de la pata (punto final). Esto implica resolver un conjunto de ecuaciones que relacionan las coordenadas del extremo de la pata con los ángulos de las articulaciones.

Se estudiará la cinemática inversa de una de las patas del robot hexápodo, abordándola de manera similar a como se trataría la cinemática inversa de un brazo manipulador. Este enfoque se utiliza debido a la analogía existente entre los movimientos de las patas del hexápodo y los de un brazo robótico. Para simplificar el análisis, se comenzará estudiando la cinemática inversa de un manipulador planar con dos articulaciones de revolución, ya que las ecuaciones derivadas de este caso pueden ser extrapoladas al escenario de la pata del hexápodo.

5.1.2.1 Cinemática inversa de un manipulador planar con 2 articulaciones de revolución

Considérese el siguiente manipulador planar:

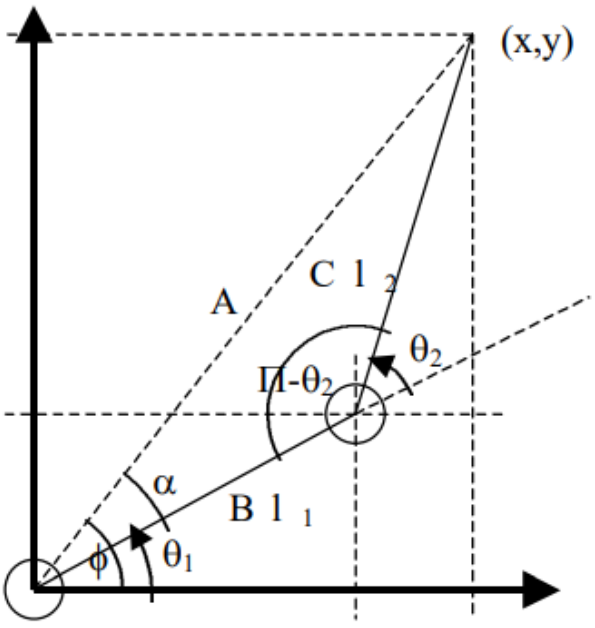


Figura 96. Manipulador planar con dos articulaciones de revolución: Fuente: [68]

Para calcular el valor de θ_2 se aplica el teorema del coseno:

$$x^2 + y^2 = {l_1}^2 + {l_2}^2 - 2l_1 l_2 \cos(\pi - \theta_2)$$

$$\cos\theta_2 = \frac{x^2 + y^2 - {l_1}^2 - {l_2}^2}{2l_1 l_2}$$

$$\theta_2 = \arccos\left(\frac{x^2 + y^2 - {l_1}^2 - {l_2}^2}{2l_1 l_2}\right)$$

θ_2 tiene dos soluciones, una positiva y otra negativa.

Para calcular el valor de θ_1 se tiene que:

$$\theta_1 = \phi - \alpha$$

$$\tan\phi = \frac{y}{x}$$

$$\tan(\alpha) = \frac{l_2 \sin\theta_2}{l_1 + l_2\cos\theta_2}$$

En función del signo que se elija para θ_2 y del valor correspondiente para θ_1 se tendrán las soluciones codo abajo y codo arriba.

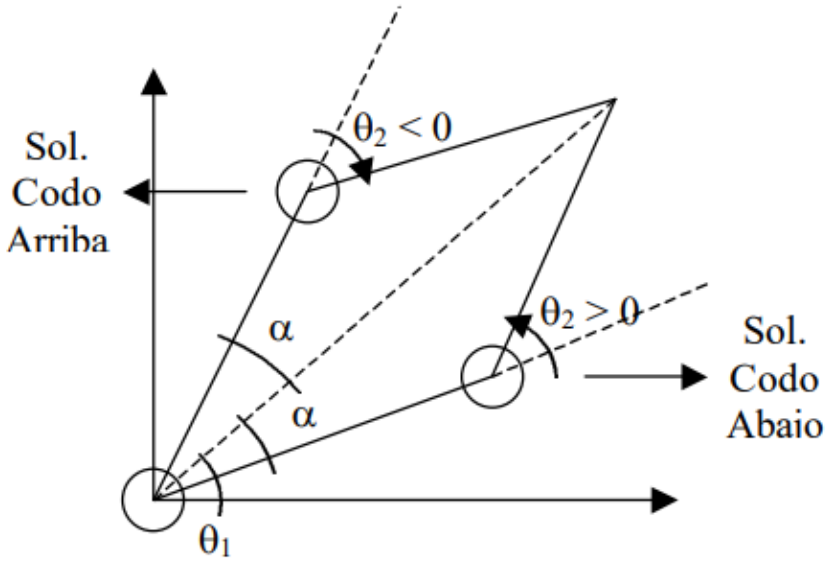


Figura 97. Configuración codo arriba y codo abajo para un manipulador planar de dos articulaciones de revolución. Fuente: [68]

5.1.2.2 Cinemática inversa de una pata del hexápodo

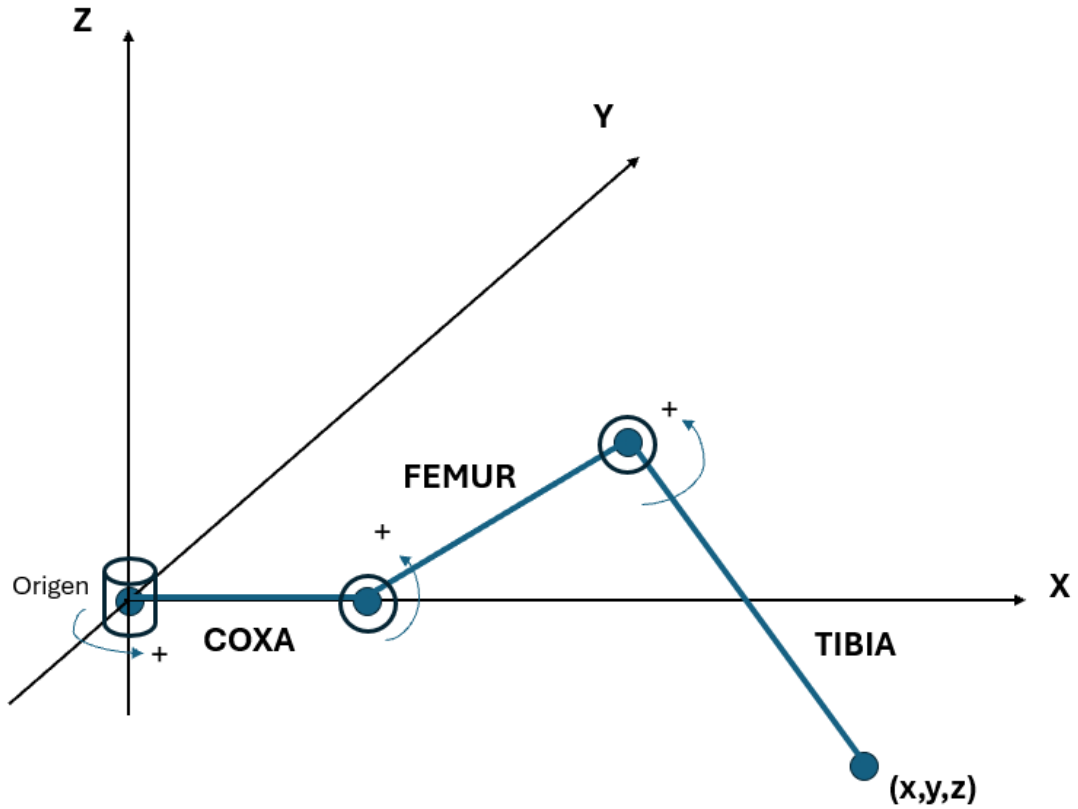


Figura 98. Representación de la pata en el espacio de coordenadas. Fuente: Elaboración propia.

En primer lugar, hallando el valor de θ_1 tenemos que:

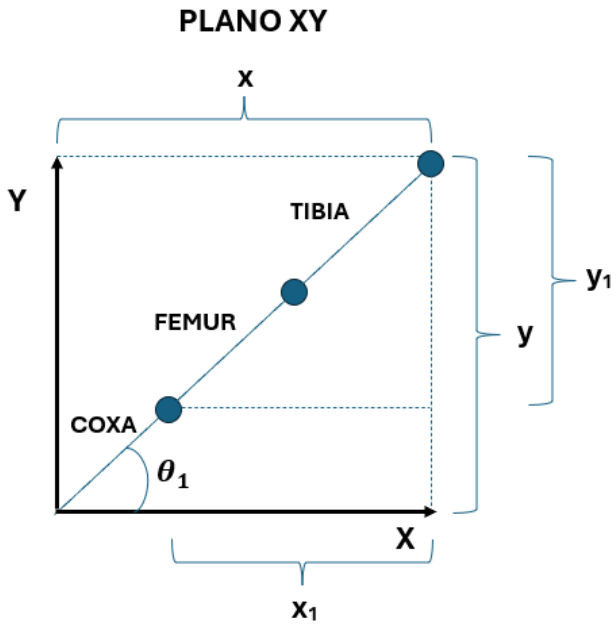


Figura 99. Representación de la pata vista desde el plano XY. Fuente: Elaboración propia.

Donde:

$$x_1 = x - l_1 \cos\theta_1$$

$$y_1 = y - l_1 \sin\theta_1$$

$$\theta_1 = arctg\left(\frac{y}{x}\right)$$

Ahora, para las demás variables articuladas y representando solamente las dos últimas articulaciones se tiene:

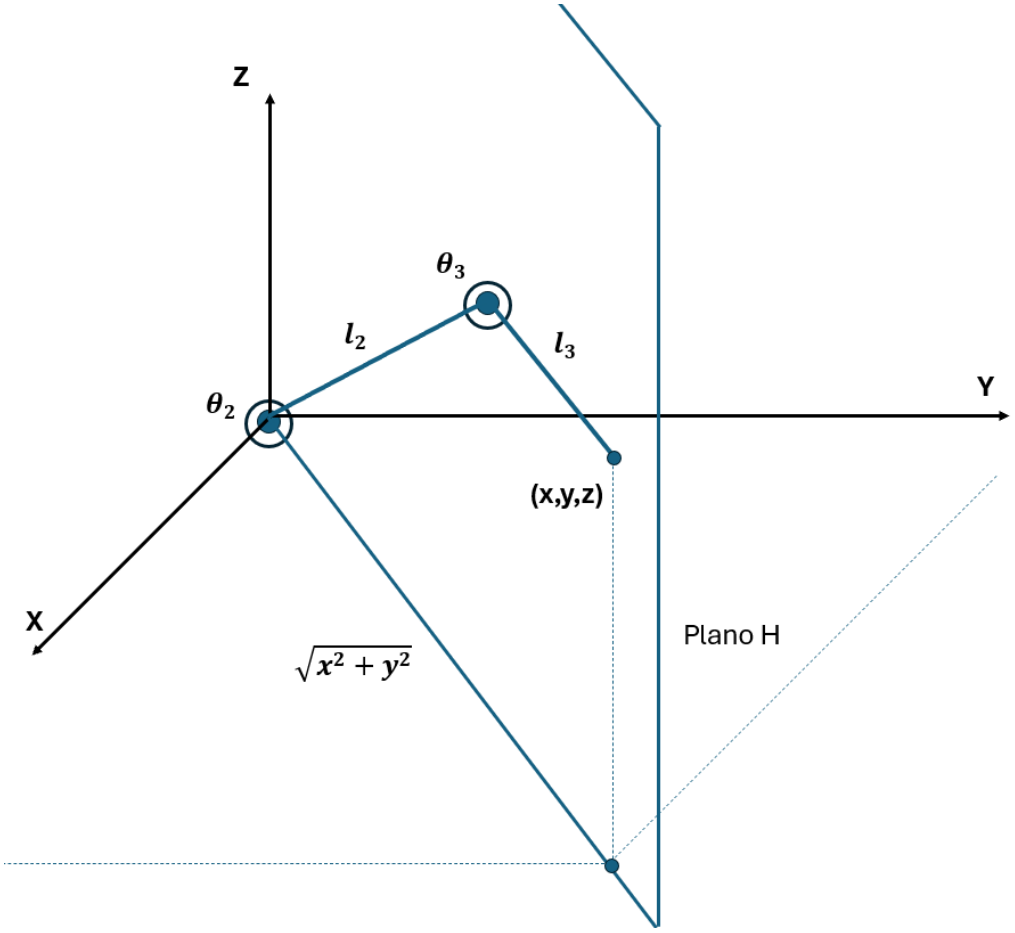


Figura 100. Representación de la pata en el espacio de coordenadas (sin la primera articulación) en el Plano H. Fuente: Elaboración propia.

Si se representa el plano H por separado, se tendrá la siguiente situación:

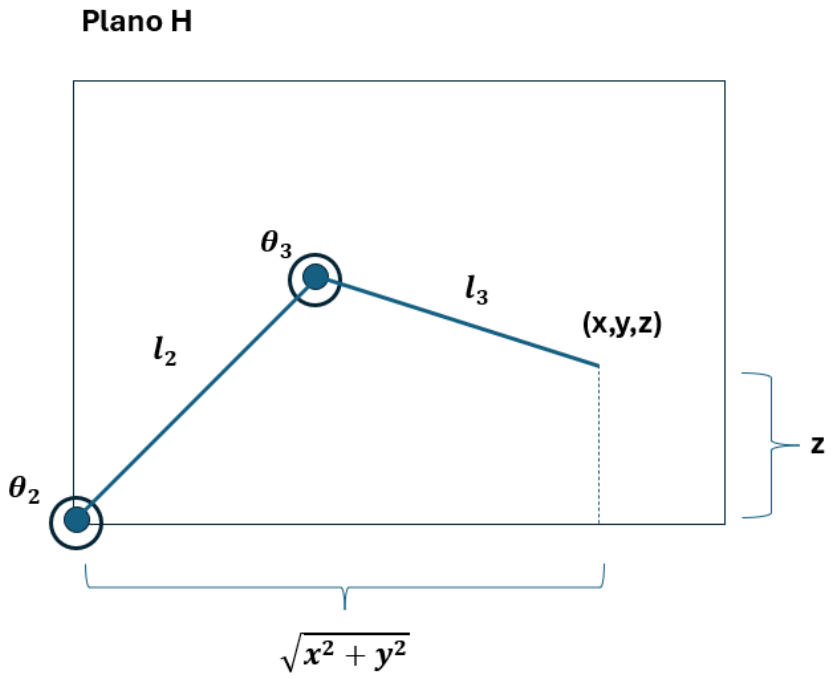


Figura 101 . Representación de la pata vista desde el plano H. Fuente: Elaboración propia.

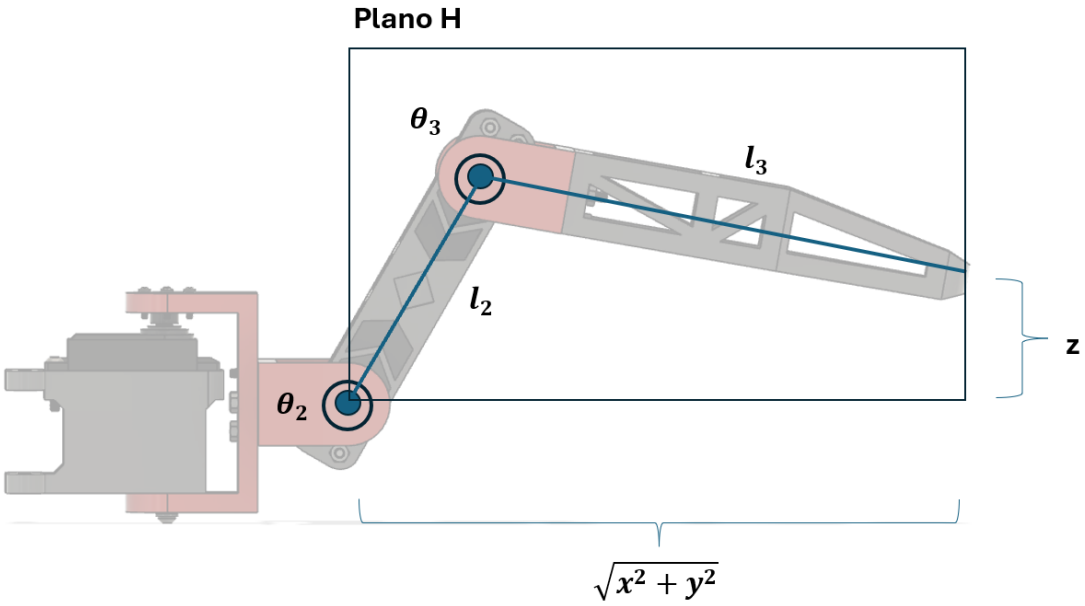


Figura 102. Representación de la pata en Fusion 360 vista desde el plano H en correspondencia con la figura superior. Fuente: Elaboración propia.

Como podemos apreciar estamos en el mismo caso que el manipulador en 2D con dos articulaciones de revolución. Por lo que procediendo de forma análoga al caso anterior y teniendo el cuanta los siguientes cambios de variables tenemos que:

Situación 2D	Situación 3D
x	$\sqrt{{x_1}^2 + {y_1}^2}$
y	z
θ_1, l_1	θ_2, l_2
θ_2, l_2	θ_3, l_3

Tabla 7. Cambio de variable para pasar del caso de un manipulador planar de dos articulaciones al caso de una pata del robot hexápodo.

Si se sustituyen las variables, de acuerdo con la tabla anterior, en las fórmulas que se habían obtenido para el caso 2D, se tiene:

$$\cos\theta_3 = \frac{{x_1}^2 + {y_1}^2 + z^2 - {l_2}^2 - {l_3}^2}{2 l_2 l_3}$$

Para codo arriba:

$$\theta_3 = -\text{aco s}\left(\frac{{x_1}^2 + {y_1}^2 + z^2 - {l_2}^2 - {l_3}^2}{2 l_2 l_3}\right)$$

Para $\theta_2$se tiene que:

$$\theta_2 = \phi - \alpha$$

$$\tan\phi = \frac{z}{\sqrt{{x_1}^2 + {y_1}^2}}$$

$$\tan\alpha = \frac{l_3 \sin\theta_3}{l_2 + l_3 \cos\theta_3}$$

5.2 Control cinemático del robot hexápodo

Una vez obtenido el modelo cinemático del robot, se busca poder establecer estrategias adecuadas de control que redunden en una mayor calidad de sus movimientos. El control cinemático establece cuáles son las trayectorias que debe seguir cada articulación del robot a lo largo del tiempo para lograr los objetivos fijados por el usuario (punto de destino, trayectoria cartesiana del efector final, tiempo invertido, etc.) [51].

5.2.1 Funciones del control cinemático

El control cinemático de un robot, ilustrado esquemáticamente en la Figura 103, recibe datos de movimiento del programa del robot escrito por el usuario. Estos datos incluyen puntos de destino, precisión, tipo de trayectoria deseada, velocidad, tiempo invertido, entre otros.

Basándose en el modelo cinemático del robot, el control cinemático establece las trayectorias para cada articulación como funciones del tiempo. Estas trayectorias se muestrean con un período T específico, generando en cada instante kT un vector de referencias para los algoritmos de control dinámico [51].

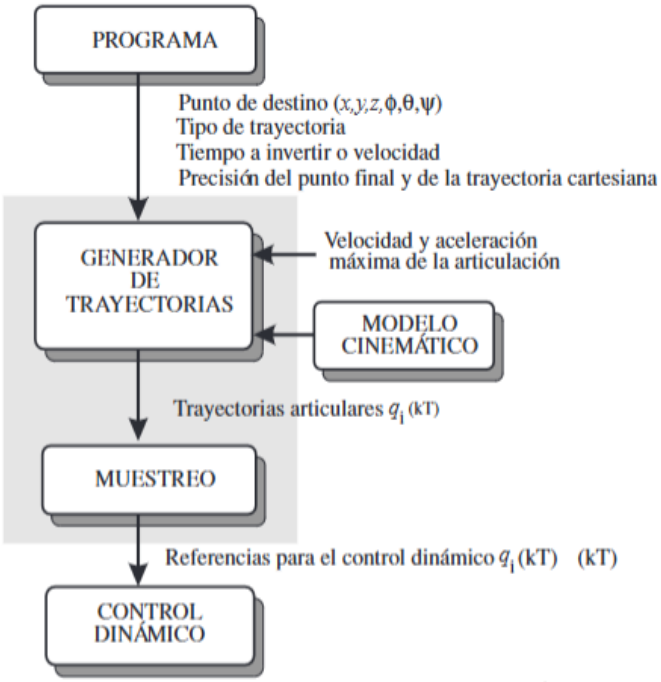


Figura 103. Funcionamiento del control cinemático (sombreado). Fuente: [51]

De manera general, el control cinemático debe realizar las siguientes funciones:

1. Interpretar la especificación del movimiento del programa mediante una trayectoria analítica en espacio cartesiano, que evoluciona dicha coordenada cartesiana en función del tiempo.
2. Mostrar la trayectoria cartesiana obtenida en puntos finitos, con cada punto dado por una tupla en nuestro caso de tres elementos (x, y, z).
3. Usando la transformación homogénea inversa, convertir estos puntos en coordenadas articulares (q_1, q_2, q_3), teniendo en cuenta posibles soluciones múltiples o ausencia de solución.
4. Interpolar los puntos articulares generando una expresión $q_i(t)$ para cada articulación, creando una trayectoria realizable que se aproxime a la especificada por el usuario.
5. Muestrear la trayectoria articular para generar referencias al control dinámico.

En algunos casos específicos, algunas funciones pueden omitirse. En nuestro caso, no se utiliza el control dinámico. Por lo tanto, las referencias generadas a partir de la trayectoria articular solo se utilizan para la planificación y control de la trayectoria, sin entrar en los detalles del control dinámico que ajustarían las fuerzas y momentos necesarios para mover el robot según esa trayectoria.

5.2.2 Generación de trayectorias cartesianas

Normalmente, el usuario del robot especifica el movimiento que este debe realizar indicando las localizaciones espaciales por las que debe pasar el extremo del robot, junto con otros datos como los instantes de paso, velocidades o tipos de trayectorias.

Por ejemplo, es común especificar que el robot debe moverse de un punto inicial a un punto final, siguiendo una línea recta a velocidad constante en el espacio de la tarea.

En ocasiones, el usuario precisa especificar una secuencia de localizaciones por las que desea que pase el extremo del robot. En estos casos, es necesario establecer un interpolador entre las localizaciones expresadas en el espacio de la tarea, que dará como resultado una expresión analítica de la evolución de cada coordenada.

La interpolación más sencilla consiste en la interpolación lineal entre dos localizaciones sucesivas, de modo que en el intervalo entre cada dos localizaciones, cada coordenada en el espacio de la tarea evoluciona a velocidad constante desde su valor inicial j^i hasta el final j^f:

$$j(t) = \left(j^f - j^i\right)\frac{t - t_i}{t_f - t_i} + j^i$$

Donde t_i y t_f son los instantes de tiempo en los que se pretende alcanzar la localización inicial y final del intervalo, respectivamente.

Este tipo de evolución causa discontinuidades en la velocidad y aceleración en los puntos de paso, por lo que su uso no es recomendable [51].

Para evitar estas discontinuidades, utilizamos técnicas de interpoladores a tramos y otros interpoladores descritos en los siguientes apartados.

5.2.3 Interpolación de trayectorias

Una vez que se dispone de la secuencia de configuraciones articulares por las que debe pasar el robot, la función siguiente del control cinemático consiste en unir esta sucesión de puntos articulares. De esta manera, se consigue una trayectoria realizable y suficientemente suave.

Para ello, se debe seleccionar algún tipo de función (frecuentemente polinómica) cuyos parámetros o coeficientes se ajusten al imponer las condiciones de contorno: posiciones, velocidades y aceleraciones.

Por ejemplo, si se desea pasar por n puntos, con la condición de que se parta y se llegue al reposo (velocidad nula), podría considerarse como primera opción utilizar un polinomio de grado $n+1$ cuyas $n+2$ condiciones se calcularían para garantizar las $n+2$ condiciones de contorno (n puntos de paso y 2 velocidades). Sin embargo, el uso de un polinomio de grado elevado presenta múltiples inconvenientes desde el punto de vista computacional. Por ejemplo, precisa resolver un sistema de ecuaciones con $n+2$ incógnitas y la presencia de potencias elevadas puede acarrear errores significativos en los cálculos.

Por estos motivos, en lugar de usar una sola función interpoladora que una todos los puntos de la trayectoria, se utilizan conjuntos de funciones más simples que interpolan localmente la secuencia de puntos, es decir, unen pocos puntos consecutivos de un intervalo, solapándose unas con otras para garantizar la continuidad [51].

A continuación, se presentarán las funciones interpoladoras utilizadas.

Cabe indicar que, aunque las técnicas de interpolación que se describirán están planteadas para el espacio articular, son igualmente aplicables para el espacio de la tarea (de hecho, de esta última manera las hemos implementado de manera software).

5.2.3.1 Interpolador spline cúbico para el "swing"

Para asegurar que la trayectoria que une los puntos por los que debe pasar la articulación del robot tenga continuidad en la velocidad, se puede utilizar un polinomio de grado 3 que una cada pareja de puntos adyacentes.

Se consigue así una trayectoria compuesta por una serie de polinomios cúbicos, cada uno válido entre dos puntos consecutivos. Este conjunto de polinomios concatenados, escogidos de modo que exista continuidad en la posición y velocidad, se denominan splines (cúbicos, por ser de tercer grado) [51].

La expresión de la trayectoria que une dos puntos adyacentes (q^i, q^{i+1}) será:

$$q(t) = a + b\left(t - t^i\right) + c\left(t - t^i\right)^2 + d\left(t - t^i\right)^3 \qquad t^i < t < t^{i+1}$$

$$a = q^i$$

$$b = \dot{q}^i$$

$$c = \frac{3}{T^2}\left(\dot{q}^{i+1} - \dot{q}^i\right) - \frac{1}{T}\left(\dot{q}^{i+1} + 2\dot{q}^i\right)$$

$$d = -\frac{2}{T^3}\left(\dot{q}^{i+1} - \dot{q}^i\right) + \frac{1}{T^2}\left(\dot{q}^{i+1} + \dot{q}^i\right)$$

$$T = t^{i+1} - t^i$$

Para calcular los valores de los coeficientes del polinomio cúbico de la expresión anterior, es preciso conocer los valores de las velocidades de paso $\dot{q}$. Para ello existen diferentes alternativas.

Un primer criterio para seleccionar las velocidades de paso podría ser:

$$\dot{q}^i = \begin{cases} 0 & \text{si signo}\left(q^i - q^{i-1}\right) \neq \text{ signo}\left(q^{i+1} - q^i\right) \\ \frac{1}{2}\left[\frac{q^{i+1} - q^i}{t^{i+1} - t^i} + \frac{q^i - q^{i-1}}{t^i - t^{i-1}}\right] & \text{si} \begin{cases} \text{signo}\left(q^i - q^{i-1}\right) = \text{signo}\left(q^{i+1} - q^i\right) \\ 0 \quad q^{i-1} = q^i \\ 0 \quad q^i = q^{i+1} \end{cases} \end{cases}$$

Esta selección es sencilla y da como resultado una continuidad razonable en la velocidad, pero no establece condiciones sobre la continuidad de la aceleración.

Como alternativa, se pueden escoger las velocidades de paso de modo que cada spline cúbico sea continuo en posición, velocidad y aceleración con los polinomios adyacentes. De esta forma, los coeficientes de los $k-1$ polinomios de tipo spline cúbico que pasan por los puntos q^i ([1, k]) asegurarán la continuidad en posición, velocidad y aceleración de la trayectoria global, y se obtendrán resolviendo el siguiente sistema de ecuaciones lineales de diagonal dominante:

$$\begin{bmatrix} t^3 & 2(t^2+t^3) & t^2 & 0 & 0 & \dots \\ 0 & t^4 & 2(t^3+t^4) & t^3 & 0 & \dots \\ 0 & 0 & t^5 & 2(t^4+t^5) & t^4 & \dots \\ \dots & \dots & 0 & t^6 & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \cdot \begin{bmatrix} \dot{q}^1 \\ \dot{q}^2 \\ \dot{q}^3 \\ \vdots \\ \dot{q}^k \end{bmatrix}$$

$$= \begin{bmatrix} \frac{3}{t^2 t^3}\left[(t^2)^2(q^3 - q^2) + (t^3)^2(q^2 - q^1)\right] \\ \frac{3}{t^3 t^4}\left[(t^3)^2(q^4 - q^3) + (t^4)^2(q^3 - q^2)\right] \\ \vdots \\ \frac{3}{t^{k-1} t^k}\left[(t^{k-1})^2(q^k - q^{k-1}) + (t^k)^2(q^{k-1} - q^{k-2})\right] \end{bmatrix}$$

Para completar el número de ecuaciones y que el sistema esté definido, se pueden añadir las siguientes condiciones:

$$\dot{q^1} = \dot{q^k} = 0$$

Esto asegura que la articulación parte y llega a una situación de reposo. Por lo tanto, las k ecuaciones lineales definidas permiten obtener las k velocidades de paso necesarias para aplicar la expresión principal de los splines cúbicos, asegurando la continuidad hasta la segunda derivada de la trayectoria global [51].

El script que desarrollamos implementa el método de condiciones de contorno naturales. Por lo que no aplicaremos el método con condiciones de frontera no nulas definiendo los valores específicos para las derivadas en los extremos ni el métodos de ajuste de velocidad ajustando las velocidades en los puntos intermedios.

5.2.3.2 Interpolador trapezoidal para el "stance"

En el interpolador vistos hasta el momento, se utiliza un polinomio de un grado determinado 3 para unir dos puntos consecutivos de la trayectoria.

El uso de polinomios de tercer grado permite asegurar que el polinomio pasa por los dos puntos y, al mismo tiempo, permite imponer los valores de velocidad de paso por los mismos.

Sin embargo, a diferencia del interpolador de primer grado (lineal), la velocidad de la articulación varía continuamente durante el recorrido, lo que exige un control continuo de la misma.

Una alternativa que proporciona una solución intermedia consiste en descomponer en tres tramos consecutivos la trayectoria que une dos puntos q^0, q^1.

En el tramo central se utiliza un interpolador lineal, manteniendo la velocidad constante y sin precisar imprimir aceleración al actuador.

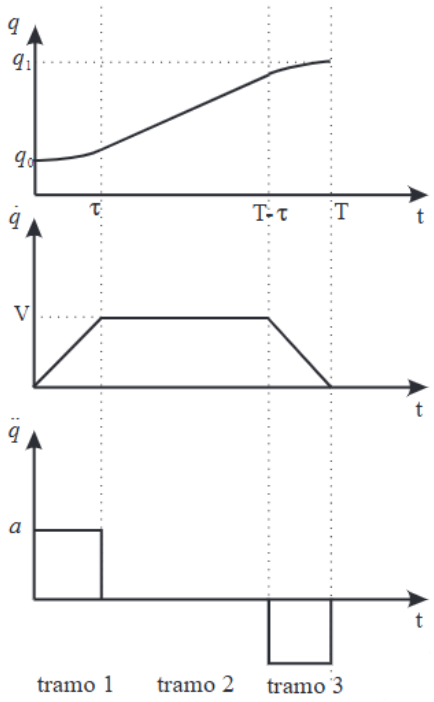


Figura 104. Perfiles de posición, velocidad y aceleración en el interpolador trapezoidal. Fuente: [51]

En los tramos inicial y final se utiliza un polinomio de segundo grado, variando linealmente la velocidad desde la de la trayectoria anterior hasta la de la presente en el tramo 1, y en el tramo 3, variando linealmente desde la velocidad de la trayectoria presente hasta la de la siguiente. Esto significa que en los tramos inicial y final, la aceleración toma valores constantes distintos de cero, mientras que en el tramo intermedio, la aceleración es nula.

En el caso simple de una trayectoria con dos únicos puntos de velocidad inicial y final nula, las ecuaciones de los tres tramos serían:

$$q(t) = \begin{cases} q^0 + s\frac{a}{2}t^2 & t \leq \tau \\ q^0 - s\frac{V^2}{2a} + sVt & \tau < t \leq T - \tau \\ q^1 + s\left(-\frac{aT^2}{2} + aTt - \frac{a}{2}t^2\right) & T - \tau < t < T \end{cases}$$

Siendo:

$$\tau = \frac{V}{a}$$
$$T = s\frac{q^1 - q^0}{V} + \frac{V}{a}$$

V : velocidad máxima permitida
a : aceleración máxima permitida
s: signo $(q^1 - q^0)$

Como se observa, el perfil de velocidad toma la forma de un trapecio, por lo que se denomina a este tipo de interpolador trapezoidal. La aceleración, por su parte, presenta discontinuidades (escalones).

Esta trayectoria responde a la de tiempo mínimo, con las restricciones de velocidad y aceleración máxima permitida, equivalentes a acelerar el movimiento lo más rápido posible, mantenerlo el tiempo necesario y desacelerarlo de nuevo tan rápido como se pueda.

Puede ocurrir que los valores q^0, q^1, V y a impidan alcanzar la velocidad V o incluso que no se llegue a alcanzarla, debiendo empezar la fase de deceleración antes de alcanzarse V. En este caso, el perfil de velocidad tiene una forma triangular.

En una trayectoria formada por varios puntos, la velocidad de paso por los puntos intermedios no debería ser nula, ya que esto causaría movimientos discontinuos del robot. Esta situación se puede evitar permitiendo que la trayectoria no pase exactamente por los puntos. La trayectoria final coincidirá con las trayectorias rectilíneas que unen los puntos, salvo en las cercanías de estos, donde un polinomio de segundo grado (aceleración constante) variará progresivamente la velocidad, evitando los valores infinitos de la aceleración.

Esta técnica, conocida como ajuste parabólico, aproxima al interpolador lineal cuanto mayor sea la aceleración permitida. Así, si se desea pasar por los puntos q^0, q^1, q^2 en los instantes $t = 0, t = T_1, t = T_1 + T_2$, las ecuaciones de los tres tramos que componen la trayectoria que une dos puntos consecutivos serían:

$$q(t) = \begin{cases} q^0 + \dfrac{q^1 - q^0}{T_1} t & 0 \le t \le T_1 - \tau \\ q^1 + \dfrac{(q^1 - q^0)}{T_1}(t - T_1) + \dfrac{a}{2}(t - T_1 + \tau)^2 & T_1 - \tau < t < T_1 + \tau \\ q^1 + \dfrac{q^2 - q^1}{T_2}(t - T_1) & T_1 + \tau < t < T_1 + T_2 \end{cases}$$

donde:

$$a = \frac{T_1(q^2 - q^1) - T_2(q^1 - q^0)}{2T_1T_2\tau}$$

El tiempo utilizado en variar la velocidad del movimiento, 2τ, se reparte simétricamente respecto al instante T_1 . Lógicamente, cuanto mayor sea la aceleración permitida, menor será el tiempo de transición τ.

5.2 Implementación de scripts en Python

El código desarrollado se compone de 2 niveles. La parte más elemental son las herramientas desarrolladas, aquí se incluyen los interpoladores, códigos de cálculo cinemático y código para procesar datos. Estas herramientas son posteriormente empleadas en otros scripts para generar las trayectorias coordinarlas y guardarlas para su posterior empleo en el ESP32.

5.2.1 Herramientas desarrolladas

5.2.1.1 Script de Cinemática directa (`direct_kinematics.py`)

La función de este script es servir como herramienta de depuración, fue especialmente útil en las etapas más tempranas del desarrollo para la comprobación de las trayectorias generadas. Se trata de un código sencillo fácilmente modificable y adaptable a diferentes dimensiones de la pata, como argumentos recibe los 3 ángulos de una pata y devuelve un vector que contiene la posición x y z del efector final de la pata. Para este cálculo se emplea Denavit – Hartenberg (D-H), método ya explicado en el Anexo 10.4.1.3 . Consultar Anexo 10.8.2 para código completo.

5.2.1.2 Script de Cinemática inversa (`inverse_kinematics.py`)

Se trata de uno de los script fundamentales y más empleado del código, su función es recibir las coordenadas x y z del efector final de la pata y devolver los ángulos necesarios

de las tres articulaciones para que esta condición se cumpla. El código asume siempre una configuración de codo arriba para alcanzar el punto con el efector final.

Para el cálculo de una trayectoria, este script se ejecutará sobre cada punto devuelto por los interpoladores desarrollados.

5.2.1.3 Script para el procesador de información (`process_data.py`)

Otro script enfocado en depuración, útil para verificar los perfiles de velocidad de los servomotores y visualizar trayectorias. Este script incluye dos funciones: `grafica_q()`, que toma como argumento los ángulos de una trayectoria para calcular la derivada aproximada mediante diferencias entre valores consecutivos, mostrando como resultado gráficos de posición, velocidad y aceleración de la pata.

Y `grafica_p()` que recibe los puntos de la trayectoria en el espacio de trabajo y devuelve la posición x y z frente al tiempo y muestra a continuación la trayectoria en el espacio de trabajo. Consultar Anexo 10.8.3 para visualizar el código.

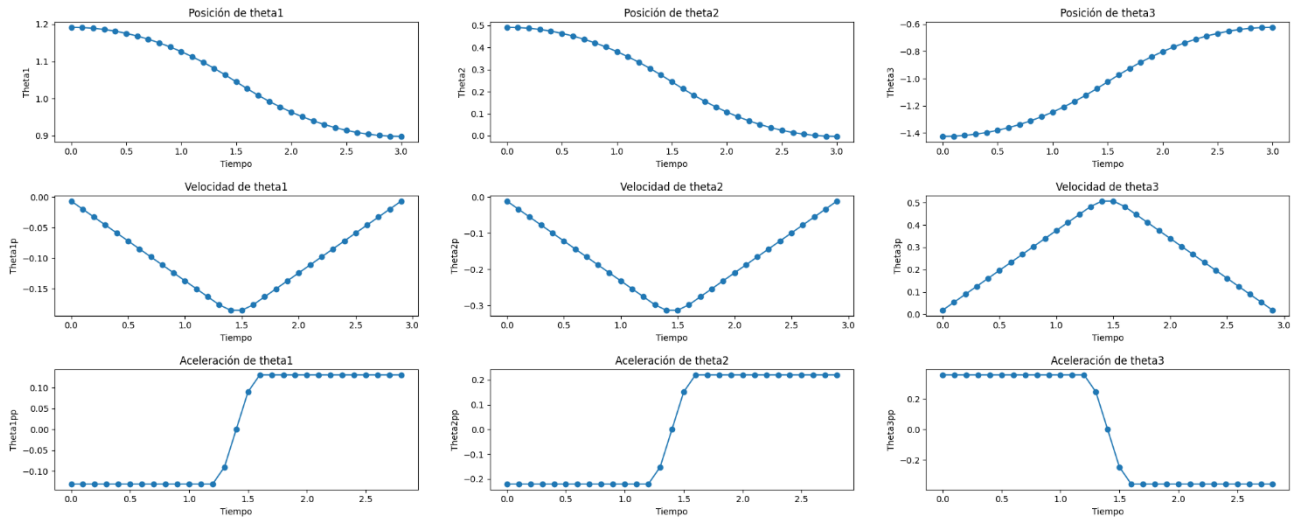


Figura 105. Graficas generadas por grafica_q() en una trayectoria con interpolador trapezoidal. Fuente: Elaboración propia.

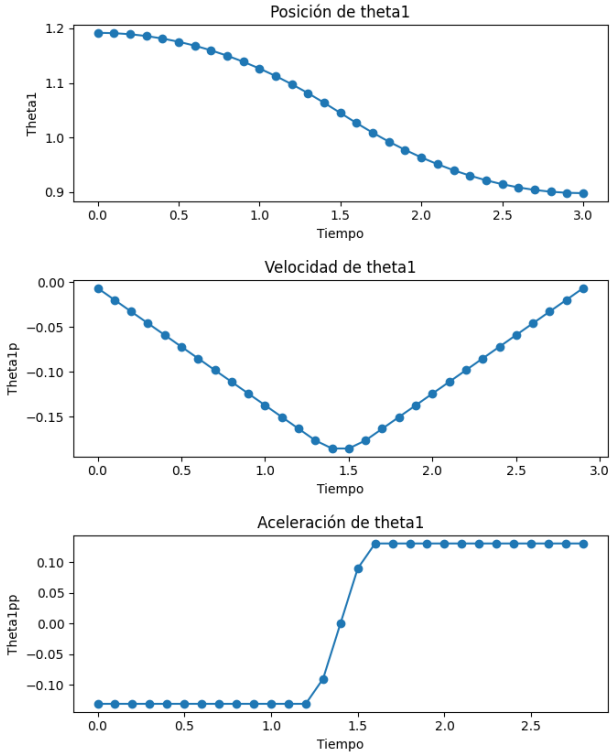


Figura 106. Zoom sobre la primera articulación de las gráficas generadas por grafica_q(). Fuente: Elaboración propia.

5.2.1.3 Interpolador spline cubico (`spline_cubico.py`)

Empleado para generar los balanceos del paso. En este script se define una función que recibe como argumentos el punto inicial, punto de paso, que se sitúa en el vértice de la parábola generada, y punto final de la trayectoria y el tiempo que se desea emplear. Para la implementación de este spline cúbico se han tomado como condiciones de frontera velocidades igual a 0 en los extremos, lo cual concuerda con el comportamiento esperado en el ciclo de un paso. Código completo en Anexo 10.8.5.

La definición del polinomio cúbico está implícita en la función `spline(t, i)` del script, que evalúa el polinomio cúbico dado los coeficientes a_i, b_i, c_i, y d_i.

```
40 def spline(t, i):
41     dt = t - tiempos[i]
42     return a[i] + b[i] * dt + c[i] * dt**2 + d[i] * dt**3
```

Las condiciones de contorno se establecen en la matriz A y el vector b:

```
14 A[0, 0] = 1
15 A[-1, -1] = 1
16 b[0] = [0, 0, 0]
17 b[-1] = [0, 0, 0]
```

Las ecuaciones de continuidad de la primera y segunda derivada en los puntos intermedios se traducen en un sistema lineal.

La construcción de la matriz A y el vector b se realiza en el bucle `for` del script:

```
19 for i in range(1, n):
20     A[i, i-1] = h[i-1]
21     A[i, i] = 2 * (h[i-1] + h[i])
22     A[i, i+1] = h[i]
23     b[i] = 3 * ((puntos[i+1] - puntos[i]) /
24                 h[i] - (puntos[i] - puntos[i-1]) /
25                 h[i-1])
```

Aquí, $h[i] = t_{i+1} - t_i$, y A y b se llenan según las derivadas y diferencias entre puntos.

A continuación el sistema lineal Ac = b se resuelve para obtener los coeficientes c_i usando `np.linalg.solve`:

```
29 c = np.linalg.solve(A, b)
```

Una vez obtenidos los c_i, se pueden calcular los demás coeficientes a_i, b_i y d_i para cada tramo del spline usando otro bucle `for`:

```
31 a = puntos[:-1]
32 b = np.zeros((n, 3))
33 d = np.zeros((n, 3))
34
35 for i in range(n):
36     b[i] = (puntos[i+1] - puntos[i]) / h[i] - h[i] * (
37             2 * c[i] + c[i+1]) / 3
38     d[i] = (c[i+1] - c[i]) / (3 * h[i])
```

5.2.1.4 Interpolador trapezoidal (`trapezoidal_interpolation.py`)

Otro script clave para el correcto funcionamiento del hexápodo.

Empleado para generar los puntos de la trayectoria que seguirá la pata. Este script cuenta con una función que recibirá como argumento el punto inicial y final de la trayectoria, el tiempo actual y tiempo total de la trayectoria. Con estos valores se calculan los parámetros tau (o blend time), V y a del interpolador y se ejecuta la interpolación 3 veces, una vez por cada articulación. Debido a que este interpolador une dos puntos mediante una línea recta se ha empleado para las trayectorias de arrastre del hexápodo. Código completo en Anexo 10.8.4.

A continuación, se muestra la definición de la velocidad máxima del tramo. A la cual se le aplica un escalar que debe estar comprendido entre 1 y 2 para que el movimiento sea posible. A mayor valor más tiempo de aceleración habrá en el recorrido obtenido en el caso limite un perfil triangular d velocidades. En nuestro caso hemos elegio acercarnos a

ese caso limite tras realizar pruebas en el laboratorio y comparar las suavidades del tramo. [[52]

```
10      difTh = thetasf - thetas0
11      V = 1.9 * difTh / T
```

Esta ecuación responde a la siguiente desigualdad, obtenida para respetar el tiempo de transición o "blend time" tau: 0 < tau < tf /2:

$$\frac{q_f - q_0}{t_f} \leq V \leq \frac{2(q_f - q_0)}{t_f}$$

Para el cálculo de `tau` y `a` se utiliza una máscara `mask` para identificar elementos de `V` que no son cero y calcular los valores de `tau` y `a` en aquellos elementos que si varían en el tiempo, de lo contrario obtendríamos un error y calculo al dividir entre V = 0 en alguno tiempos.

```
12 tau = np.zeros_like(V)
13 a = np.zeros_like(V)
14
15 mask = V != 0
16 tau[mask] = (thetas0[mask] - thetasf[mask] + V[mask] * T) / V[mask]
17
18 a[mask] = V[mask] / tau[mask]
```

Finalmente, la función `interpolation` calcula la posición `theta_t` en función del tiempo `t` para los tres tramos del interpolador trapezoidal:

```
20 def interpolation(th0, thf, V, a, tau, t, T):
21     if th0 == thf:
22         return th0
23     if t <= tau:
24         theta_t = th0 + a / 2 * t**2
25     if tau < t <= (T - tau):
26         theta_t = (thf + th0 - V * T) / 2 + V * t
27     if (T - tau) < t <= T:
28         theta_t = (thf - a / 2 * T**2 + a * T * t -
29                    a / 2 * t**2)
30     return theta_t
```

Hay que señalar que en el script se utiliza un bucle `while` para evaluar la función `interpolation` para cada una de las tres articulaciones. Las posiciones calculadas se almacenan en el array `theta_t`:

```
32 i = 0
33 while i < 3:
34     temp = interpolation(thetas0[i], thetasf[i],
35                          V[i], a[i], tau[i], t, T)
36     theta_t[i] = temp
37     i += 1
```

```
38
39 return theta_t
```

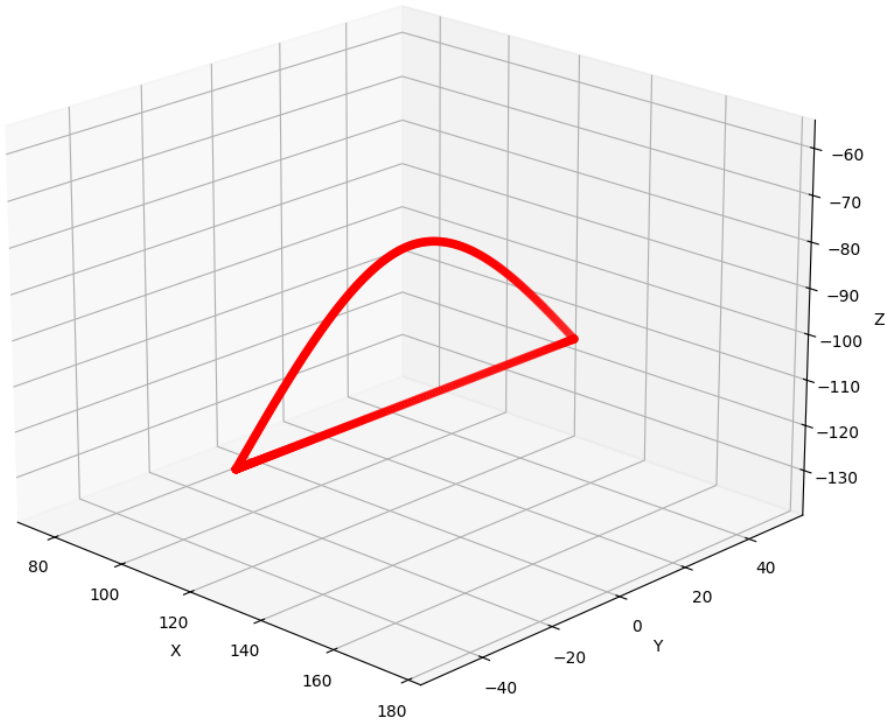


Figura. Paso obtenido mediante la suma del interpolador trapezoidal y splin-cubico usando script `process_data`. Fuente: Elaboración propia

5.2.1.5 Interpolador para la rotación (`Rotation.py`)

Para que el hexápodo pueda rotar en su posición, se ha diseñado un interpolador que conecta dos puntos mediante un arco. Esto se consigue utilizando el interpolador trapezoidal descrito anteriormente entre dos valores angulares y fijando una altura en z para la rotación. Posteriormente, estos valores angulares espaciados por el interpolador se introducirán en la ecuación de una circunferencia.

El código asume que el centro de rotación es el centro del cuerpo del hexápodo, por lo que es necesario calcular los parámetros requeridos para trasladar los puntos desde el centro del hexápodo hasta los ejes individuales de cada pata. Esto se logra midiendo la distancia entre ambos puntos de referencia y aplicando la ecuación de una circunferencia con un origen distinto al del sistema de coordenadas.

Para las patas 1, 3, 4 y 6 el centro de la circunferencia, centro del hexápodo, esté desplazado: -115.004mm en X y 11.113mm en Y.

En las patas 2 y 5 el centro de la circunferencia está desplazada: -97.32mm en X y 0mm en Y.

Por tanto, se puede aplicar la siguiente ecuación, siendo h el desfase en X y k el desfase en Y:

$$x\_points = h + r * \cos(angles\_radians)$$

$$y\_points = k + r * \sin(angles\_radians)$$

Código que realiza lo anteriormente descrito:

```
for t in np.linspace(0, T, puntos):
        angulos = trapint.calcular_qt(p0, pf, t, T,)
#hacemos la correspondencia ángulo --> punto del espacio
        x = hx + radio * np.cos(angulos[0])
        y = ky + radio * np.sin(angulos[0])
        z = altura_z
```

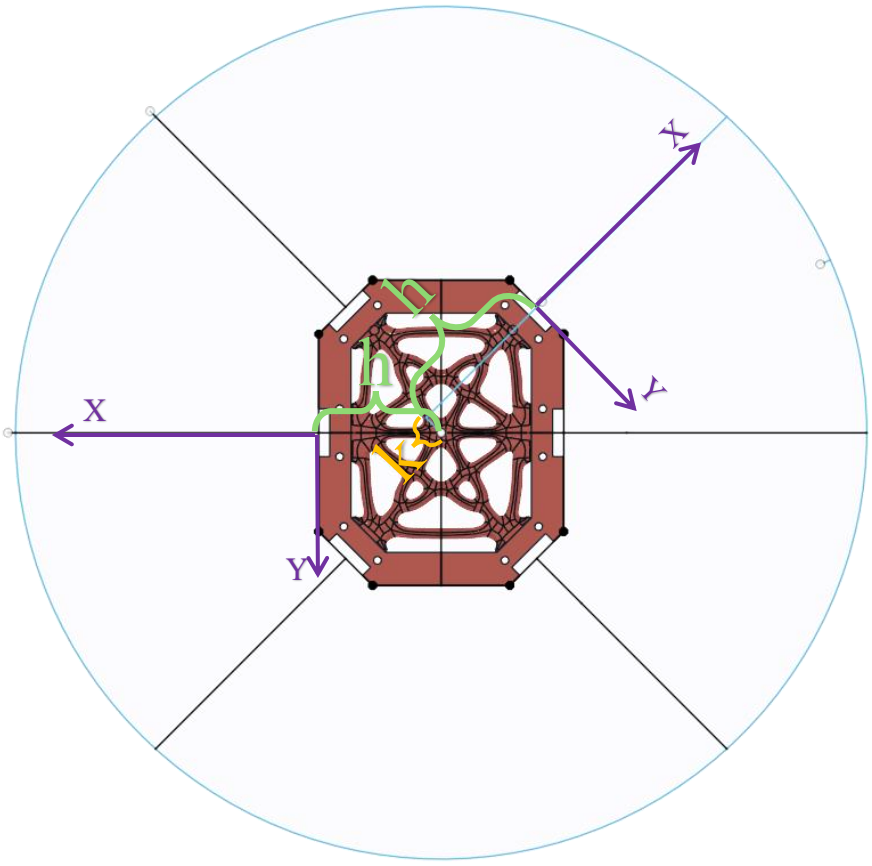


Figura 107. Desplazamientos del centro de la circunferencia que se ha de realizar. Fuente: Elaboración propia.

Por tanto, la rotación consistirá en desplazar el trípode de apoyo en una misma dirección sobre la circunferencia, mientras que el balanceo se realiza en el sentido opuesto, para así repetir el ciclo. Para consultar el código acudir al Anexo 10.8.6.

5.2.2 Planificadores de las trayectorias

Con las herramientas desarrolladas se procedió a la planificación de los modos de locomoción y coordinación de las patas. Se diseñaron dos planificadores, que operan con una misma filosofía, uno orientado al movimiento lineal del hexápodo y otro a la rotación de este. Cabe destacar que el modo de locomoción elegido para todos los desplazamientos fue la marcha en trípode ya que ofrecía el equilibrio perfecto entre estabilidad y velocidad.

Antes de empezar conviene saber cómo se sitúan los ejes x y z en nuestro hexápodo. Todas las patas se manejan de forma individual, y presentan un espacio de trabajo propio con el siguiente origen y sentido de los ejes:

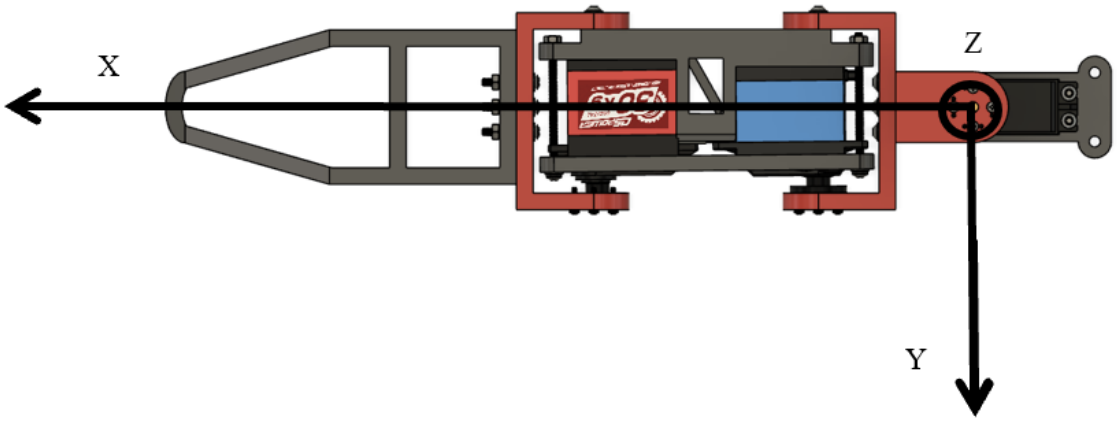


Figura 108. Ejes situados en las patas (eje Z saliente del plano). Fuente: Elaboración propia

5.2.2.1 Movimiento lineal (*movimineto_lineal.py*)

Como su nombre indica este script se encarga de generar y coordinar las trayectorias de las 6 patas para que el hexápodo se pueda mover hacia delante o hacia detrás.

En el inicio del código se definen los parámetros elementales del paso, tales como la altura del paso, distancia de paso y origen. Por último se define los tiempos empleados en las trayectorias de balanceo y arrastre:

```
#------------AJUSTAR TAMAÑO DEL PASO ---------------
#Definimos el punto de inicio y el punto final a alcanzar
#Los valores de X0 y Xf deben ser iguales
semipasoy = 50 #mm Distancia del semipaso = 2*paso
centroy = 0#mm Centro del movimineto (y reposo)
alturaz = -114#mm Nivel del suelo
separacionx = 126
#punto Reposo aprox x,y,z(126,0,-114)mm
pm = np.array([separacionx, centroy, alturaz])

#---------------- T para el paso----------
T = 2#s tiempo final
Treposo = 1#s tiempo final pasos de reposo
```

```
# Tiempos correspondientes
tiemposaire = np.array([0, T/2, T])
tiemposairereposo = np.array([0, Treposo/2, Treposo])
```

A continuación, se definen los puntos inicial y final que tendrá la trayectoria de arrastre, así como los tres puntos que deberá recorrer la pata durante el balanceo para las patas 2 y 5 (enumerando las patas en sentido de las agujas del reloj) ya que ambas se encuentran a 90º de la dirección de avance. Para la generación de estos puntos de paso se emplean los parámetros predefinidos del paso permitiendo un rápido ajuste de las trayectorias:

```
#-----PUNTOS PATAS 90º----------
#puntos para el arrastre
#punto inicial
p090 = np.array([separacionx, pm[1]-semipasoy, alturaz])
#punto final
pf90 = np.array([separacionx, pm[1]+semipasoy, alturaz])

#puntos para aire
paire90p2 = np.array([
    [pf90[0], pf90[1], pf90[2]],  # Punto inicial (x0, y0, z0)
    [pm[0], pm[1], -80],  # Punto intermedio (x1, y1, z1)
    [p090[0], p090[1], p090[2]] # Punto final (x2, y2, z2)
    ])

paire90p5 = np.array([
    [p090[0], p090[1], p090[2]],  # Punto inicial (x0, y0, z0)
    [pm[0], pm[1], -80],  # Punto intermedio (x1, y1, z1)
    [pf90[0], pf90[1], pf90[2]] # Punto final (x2, y2, z2)
    ])
```

Nótese como debido a la simetría del problema para las patas 2 y 5 los puntos de la pata 5 se obtiene invirtiendo los puntos de la pata 2.

Para las patas situadas a 45º se ha de aplicar una rotación a los puntos calculados para que estos sean paralelos a la dirección de avance, de lo contrario estas patas generaran pasos a 45º, ver Figura 109. La rotación por aplicar puede ser positiva o negativa, dependiendo de la ubicación de la pata respecto al cuerpo del hexápodo. A las patas 1 y 4 les corresponde una rotación de +45º y a las 3 y 6 de -45º. Tras esta rotación se trabaja de la misma forma que con los puntos a 90º.

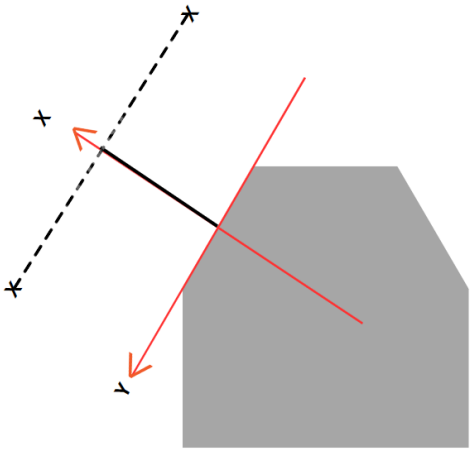


Figura 109. Ubicación de los puntos previo a la rotación de 45º para las patas 1,3,4 y 6. Fuente: Elaboración propia

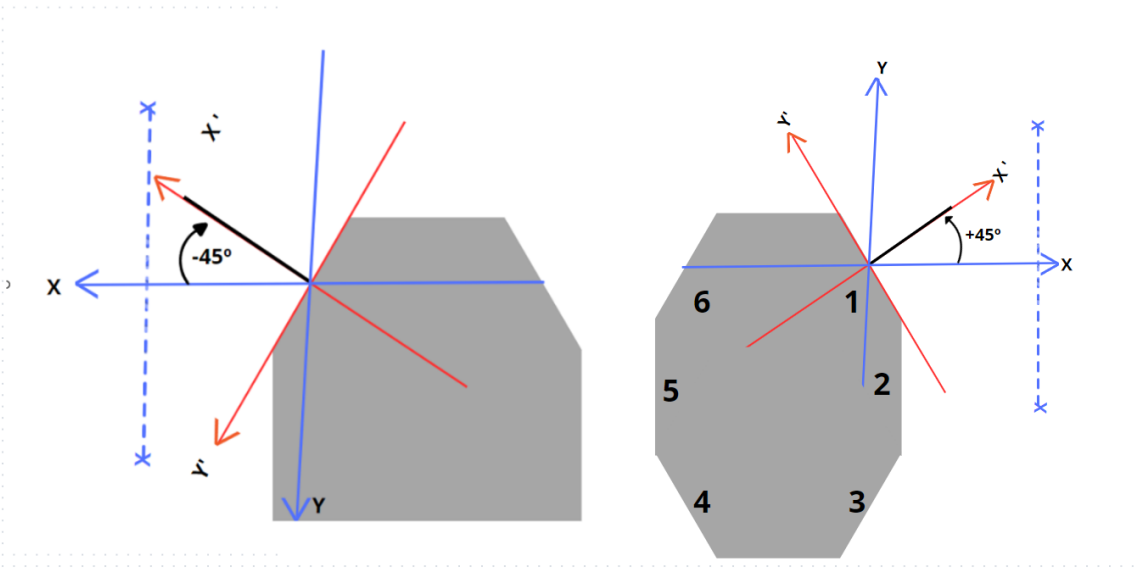


Figura 110. Izquierda: rotación negativa de los puntos. Derecha: rotación positiva de los puntos de paso. Fuente: Elaboración propia

El siguiente código aplica la rotación positiva a los puntos:

```
#----------------PUNTOS PATAS 45°--------------------
#punto inicial se aplica una rotación de +45°
p0mas45 = np.array([separacionx + semipasoy*np.cos(np.pi/4),
                    -semipasoy*np.sin(np.pi/4), alturaz])
#punto final se aplica una rotación de +45°
pfmas45 = np.array([separacionx - semipasoy*np.cos(np.pi/4),
                    +semipasoy*np.sin(np.pi/4), alturaz])


p0menos45 = np.array([separacionx - semipasoy*np.cos(np.pi/4),
                      -semipasoy*np.sin(np.pi/4), alturaz])
#punto final se aplica una rotación de -45°
```

```
pfmenos45 = np.array([separacionx + semipasoy*np.cos(np.pi/4),
                      +semipasoy*np.sin(np.pi/4), alturaz])
#punto inicial se aplica una rotación de -45°
```

Con los parámetros de la trayectoria definidos se procede a generar la trayectoria. Se generan 2 trayectorias por pata, la trayectoria de balanceo y la de arrastre. A continuación, se muestra un fragmento de código para la generación del paso para la pata 2:

```
#------------------- Angulos Arrastre (PRIMER TRAMO) PATA 2 -----
----------
poses = []
ticks_arrastre2 = []

for t in np.linspace(0, T, num=int(T*105)):
    pos_t = trapint.calcular_qt(p090, pf90, t, T)
    poses.append((pos_t, t))  # Vector xyz en el tiempo
    theta_t = ink.calcular_angulos(*pos_t)
    # Convertimos radianes a grados
    angulo_t = np.array([
        ((theta_t[0]*180.0)/np.pi) + 90,
        ((theta_t[1]*180.0)/np.pi) + 90,
        360 - ((theta_t[2]*180.0)/np.pi)
    ])
    # Corregir 3er ángulo
    if angulo_t[2] < 0:
        angulo_t[2] += 360
    if angulo_t[2] > 360:
        angulo_t[2] -= 360
    # Convertimos a grados según calibración
    us = np.array([
        (205.0/18.0)*angulo_t[0] + 1069.0/2.0,
        (313.0/30.0)*angulo_t[1] + 1187.0/2.0,
        (199.0/18.0)*angulo_t[2] + 1005.0/2.0
    ])
    # Convertimos us a ticks
    ticks_t = np.array([
        int(us[0]/4.88),
        int(us[1]/4.88),
        int(us[2]/4.88)
    ])
    ticks_arrastre2.append((ticks_t, t))

#------------------- Angulos Arrastre (PRIMER TRAMO) PATA 5 -----
----------
poses = []
ticks_arrastre5 = []

for t in np.linspace(0, T, num=int(T*105)):
    pos_t = trapint.calcular_qt(pf90, p090, t, T)  # Dirección
opuesta
    poses.append((pos_t, t))  # Vector xyz en el tiempo
    theta_t = ink.calcular_angulos(*pos_t)
    # Convertimos radianes a grados
    angulo_t = np.array([
        ((theta_t[0]*180.0)/np.pi) + 90,
```

```
172             ((theta_t[1]*180.0)/np.pi) + 90,
173             360 - ((theta_t[2]*180.0)/np.pi)
174         ])
175         # Corregir 3er ángulo
176         if angulo_t[2] < 0:
177             angulo_t[2] += 360
178         if angulo_t[2] > 360:
179             angulo_t[2] -= 360
180         # Convertimos a grados según calibración
181         us = np.array([
182             (361.0/30.0)*angulo_t[0] + 661.0/2.0,
183             (97.0/9.0)*angulo_t[1] + 550.0,
184             (524.0/45.0)*angulo_t[2] + 523.0
185         ])
186         # Convertimos us a ticks
187         ticks_t = np.array([
188             int(us[0]/4.88),
189             int(us[1]/4.88),
190             int(us[2]/4.88)
191         ])
192         ticks_arrastre5.append((ticks_t, t))
```

Si bien los puntos empleados y los interpoladores utilizados (`trapint.` trapezoidal interpolator y `sc.` splin cubico) varían, una vez generado un punto de la trayectoria, denominado `pos_t` , el procesado de los datos es el mismo:

1. Se calcula los ángulos de las 3 articulaciones para el punto `pos_t`, se emplea `ink.` (inverse_kinematics)
2. Se realiza una conversión a grados para facilitar su manipulación y se aplica un desfase. Esto se debe a que los ángulos se desplazan de manera diferente en el servomotor según su orientación. En el caso de las dos primeras articulaciones, el problema es bastante directo, ya que ambas se desplazan en el mismo sentido y solo es necesario aplicar un desfase de +90º. Ver figura 116.
3. En cuanto a la tercera articulación, es necesario no solo aplicar un desfase, sino también corregir el ángulo para que sea positivo y esté comprendido entre 0 y 360 grados.
4. Aplicamos las ecuaciones obtenidas de la recta de calibración y obtenemos la equivalencia en us para le generación de la PWM. Ver Tabla 5. Apartado 4.2.2.4.
5. Finalmente se realiza una conversión de us a ticks para su lectura por los módulos PCA9685.

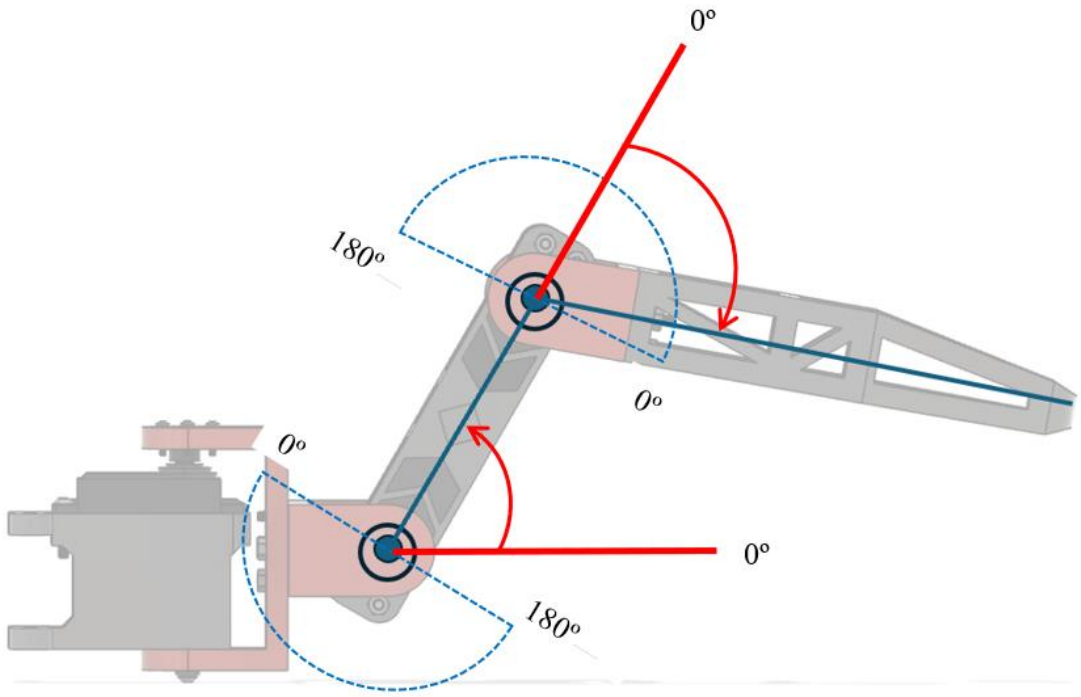


Figura 111. Desfases y sentidos de los ángulos: Rojo cálculos en Python, Azul ángulos reales. Fuente: Elaboración propia.

Para poder realizar el paso calculado de forma suave es necesario que las patas se encuentren en los puntos de inicio de la trayectoria, de lo contrario se producirá un fuerte balanceo que no seguirá ninguna trayectoria al enviar la primera triada de ángulos. Por ello de diseñan también 2 pasos más, un paso para salir del estado de reposo y otro para retornar el hexápodo al estado de reposo, especialmente útil para cambiar de dirección o de trayectoria a emplear. La generación de estos pasos es idéntica al paso en desplazamiento únicamente varía los puntos de paso que se le pasan como argumento a los interpoladores.

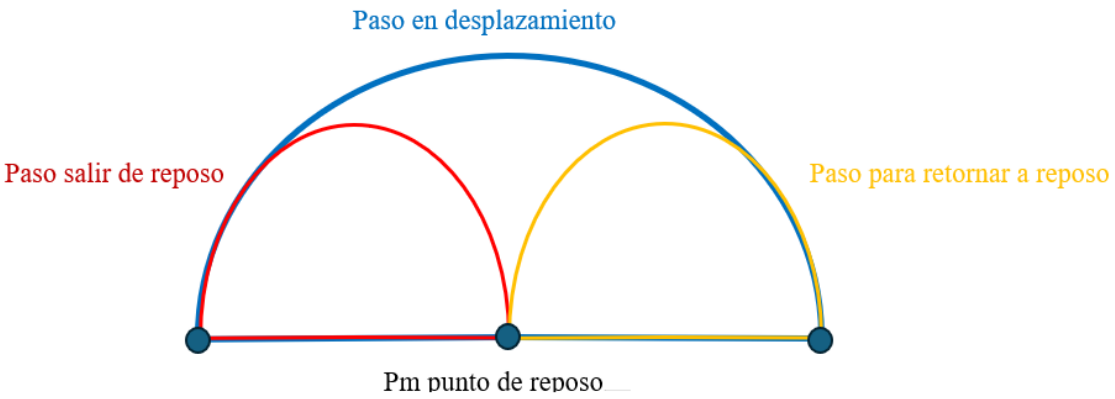


Figura 112. Pasos de reposo y retorno implementados. Fuente: Elaboración propia.

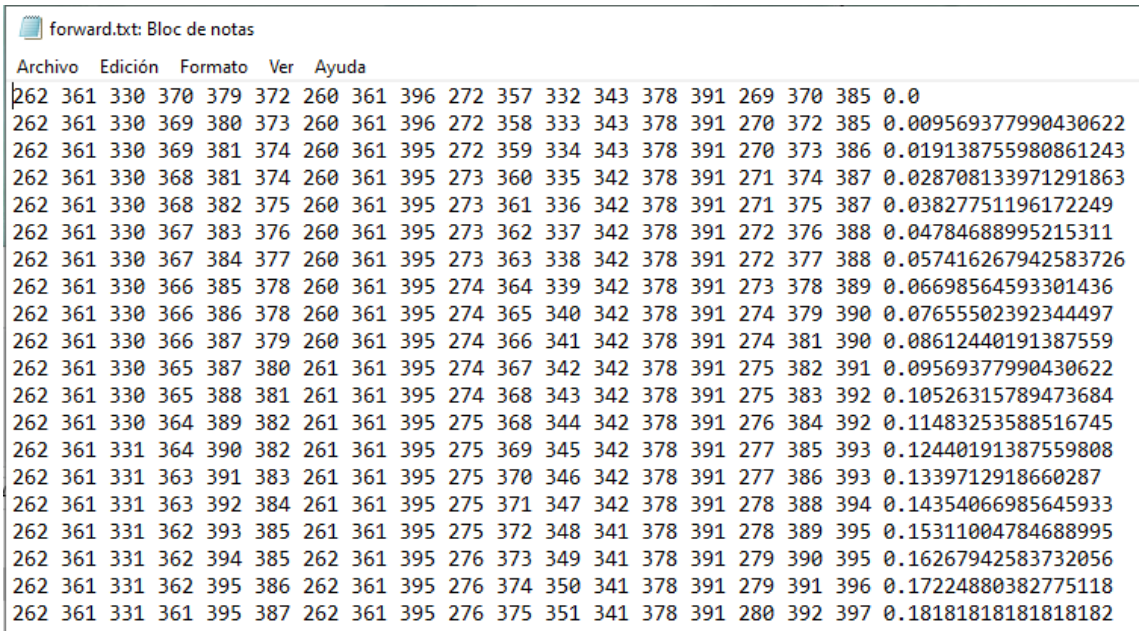
Por último se guardan los ticks generados en archivos .txt que serán almacenados y leídos por el ESP32. A continuación se muestra la función `escribir_lista_entxt()` que genera los ficheros .txt junto a una parte del código que ordena los ticks de las trayectorias en un Tripod Gait de avance hacia delante "forward" y llama a la función `escribir_lista_entxt()`. Se procede del igualmente para los pasos de retorno y reposo.

```
#-----------FUNCION PARA GUARDAR TICKS EN TXT---------------------
def escribir_lita_entxt(thetas, ruta_archivo):
    with open(ruta_archivo, 'w', newline='') as archivo:
        for array, valor_adicional in thetas:
            linea = list(array) + [valor_adicional]
            linea_texto = ' '.join(map(str,linea))
            archivo.write(linea_texto + '\n')

'''''
---GENERAR FICHERO CON LOS TICKS PARA EL PASO F Y B--------------
'''''
#-----------------PRIMER TRIPOD-----------------
thetas = []
for i, theta_t in enumerate(ticks_arrastre2):
    t = theta_t[1]
    aux = np.concatenate((ticks_arrastre1[i][0], ticks_aire2[i][0],
                          ticks_arrastre3[i][0], ticks_aire4[i][0],
                          ticks_arrastre5[i][0],ticks_aire6[i][0]))
    thetas.append((aux, t))

#-----------------SEGUNDO TRIPOD-----------------
for i, theta_t in enumerate(ticks_arrastre2):
    t = theta_t[1]
    aux = np.concatenate((ticks_aire1[i][0], ticks_arrastre2[i][0],
                          ticks_aire3[i][0], ticks_arrastre4[i][0],
                          ticks_aire5[i][0],ticks_arrastre6[i][0]))
    thetas.append((aux, t))
escribir_lita_entxt(thetas, 'forward.txt')
```

Los archivos .txt generados tendrán 18 columnas (3 articulaciones * 6 patas) y tantas filas como puntos se hayan decidido generar. La columna izquierda marcará los tiempos en los que cada conjunto de ángulos debe ser alcanzado.


forward.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
262 361 330 370 379 372 260 361 396 272 357 332 343 378 391 269 370 385 0.0
262 361 330 369 380 373 260 361 396 272 358 333 343 378 391 270 372 385 0.009569377990430622
262 361 330 369 381 374 260 361 395 272 359 334 343 378 391 270 373 386 0.019138755980861243
262 361 330 368 381 374 260 361 395 273 360 335 342 378 391 271 374 387 0.028708133971291863
262 361 330 368 382 375 260 361 395 273 361 336 342 378 391 271 375 387 0.03827751196172249
262 361 330 367 383 376 260 361 395 273 362 337 342 378 391 272 376 388 0.04784688995215311
262 361 330 367 384 377 260 361 395 273 363 338 342 378 391 272 377 388 0.057416267942583726
262 361 330 366 385 378 260 361 395 274 364 339 342 378 391 273 378 389 0.06698564593301436
262 361 330 366 386 378 260 361 395 274 365 340 342 378 391 274 379 390 0.07655502392344497
262 361 330 366 387 379 260 361 395 274 366 341 342 378 391 274 381 390 0.08612440191387559
262 361 330 365 387 380 261 361 395 274 367 342 342 378 391 275 382 391 0.09569377990430622
262 361 330 365 388 381 261 361 395 274 368 343 342 378 391 275 383 392 0.10526315789473684
262 361 330 364 389 382 261 361 395 275 368 344 342 378 391 276 384 392 0.11483253588516745
262 361 331 364 390 382 261 361 395 275 369 345 342 378 391 277 385 393 0.12440191387559808
262 361 331 363 391 383 261 361 395 275 370 346 342 378 391 277 386 393 0.1339712918660287
262 361 331 363 392 384 261 361 395 275 371 347 342 378 391 278 388 394 0.14354066985645933
262 361 331 362 393 385 261 361 395 275 372 348 341 378 391 278 389 395 0.15311004784688995
262 361 331 362 394 385 262 361 395 276 373 349 341 378 391 279 390 395 0.16267942583732056
262 361 331 362 395 386 262 361 395 276 374 350 341 378 391 279 391 396 0.17224880382775118
262 361 331 361 395 387 262 361 395 276 375 351 341 378 391 280 392 397 0.18181818181818182
```

Figura 113. Imagen recortada de un archivo .txt generado para el movimiento lineal. Fuente Elaboración propia

5.2.2.2 Giro estacionario (*Rotation parametros us.py*)

Opera de la misma forma que el script `movimiento_lineal.py`, con la única diferencia de que, en lugar de emplear un interpolador trapezoidal para el arrastre del paso, se utiliza el interpolador `Rotation.py`. Por ello, se definen nuevos parámetros, reemplazando los puntos de inicio y fin de arrastre por los ángulos de inicio a0 y fin af e introduciendo en el interpolador el desfase existente entre esa pata y el centro de rotación. Consultar código completo en el Anexo 10.8.7.

Ejemplo arrastre para la pata 2:

```
#-------------------------ROTACION ------------
#-------ARRASTRES PATA 2-----------------------
ticks_Rotarrastre2 = []
hx = -97.32
ky = 0
a0 = 15.3 * (np.pi / 180)
af = -15.3 * (np.pi / 180)
poses, Thetas_Rotarrastre2 = Rotation.semicircle_generatorxy(radio,
altura_z, puntos, T, hx, ky, a0, af)

for theta_t, t in Thetas_Rotarrastre2:
    # Pasamos los radianes a grados y consideramos el
    # desfase entre el montaje y el cálculo teorico
    angulo_t = np.array([
        ((theta_t[0] * 180.0) / np.pi) + 90,
        ((theta_t[1] * 180.0) / np.pi) + 90,
        360 - ((theta_t[2] * 180.0) / np.pi)
    ])

    # Corregir 3 angulo
    if angulo_t[2] < 0:
```

```
22          angulo_t[2] += 360
23      if angulo_t[2] > 360:
24          angulo_t[2] -= 360
25
26      # Pasamos los angulos a grados segun la recta
27      # de calibracion
28      us = np.array([
29          (205.0 / 18.0) * angulo_t[0] + 1069.0 / 2.0,
30          (313.0 / 30.0) * angulo_t[1] + 1187.0 / 2.0,
31          (199.0 / 18.0) * angulo_t[2] + 1005.0 / 2.0
32      ])
33
34      # Pasamos los us a ticks para su lectura por el
35      # modulo PCA9685
36      ticks_t = np.array([int(us[0] / 4.88), int(us[1] / 4.88),
37  int(us[2] / 4.88)])
38      ticks_Rotarrastre2.append((us.astype(int), t))
```

5.2.2.3 Pasos derivados (*Pasomodificado.py*)

El hexápodo cuenta con 2 pasos alternativos que se obtienen mediante el cambio de los parámetros base los dos planificadores de trayectorias anteriormente explicados.

El primer paso recibe el nombre de "forwardobs", obtenido al modificar el parámetro de altura de paso en el planificador de movimiento lineal. Está diseñado para ser utilizado en terrenos con obstáculos, permitiendo al robot pasar por encima de ellos con este paso.

El segundo paso se llama "forwardup", obtenido también modificando el planificador de movimiento lineal. En este caso, se reduce la altura del suelo, lo que provoca que el robot se eleve. Este paso puede ser útil en terrenos con mala visibilidad, como zonas colinosas, ya que proporciona un mejor campo de visión en caso de decidir equipar al hexápodo con una cámara o dispositivo sensorial, como un LiDAR.

5.4 Implementación del código en ESP32

El código para el ESP32 fue desarrollado en Arduino IDE y se divide en cuatro partes: declaración de variables y librerías, void setup, declaración de funciones y void loop. El código completo se encuentra en el Anexo 10.8.10.

5.4.1 Declaración de variables y librerías

En las primeras líneas de código se introducen las librerías y declaran las variables.

```
1   //variables para la lectura del BT
2   char dato ;
3   char previo = 0;
4   //variables para el cambio de modo de paso
5   char fileName1[] = "/forwardobs.txt";
6   char fileName2[] = "/reposoobs.txt";
7   char fileName3[] = "/returnobs.txt";
8
9   //Matrices que contendrán las trayectorias
10  const int rowsA = 105;
11  const int colsA = 18;
12  uint16_t matrixRP[rowsA][colsA];
13  uint16_t matrixRN[rowsA][colsA];
14  uint16_t matrixROTRP[rowsA][colsA];
15  uint16_t matrixROTRN[rowsA][colsA];
16
17  const int rows = 420;
18  const int cols = 18;
19  uint16_t matrixFW[rows][cols];
20  uint16_t matrixRT[rows][cols];
21
22  //variables empleadas para enviar los ticks a los
23  modulos
24  int P1ticks1, P1ticks2, P1ticks3;
25  int P2ticks1, P2ticks2, P2ticks3;
26  int P3ticks1, P3ticks2, P3ticks3;
27  int P4ticks1, P4ticks2, P4ticks3;
28  int P5ticks1, P5ticks2, P5ticks3;
29  int P6ticks1, P6ticks2, P6ticks3;
```

5.4.2 Void set up

En el void set up se configura las comunicaciones y se inicializa el SPIFFS, sistema de archivos empleado para la lectura de la memoria flash SPI el microcontrolador.

```
38 // ------------------------------------------------
39 void setup() {
40   Wire.begin();
41   //ajustar la velocidad de comunicación del bus I2C
42   Wire.setClock(1000000);
43   //ajustar la velocidad de comunicación del serial
44   Serial.begin(115200);
45   //Configuramos el BT y lo habilitamos
```

```
46   SerialBT.begin("HEXAPOD_ESP32");  //Bluetooth device name
47   Serial.println("BT started");
48
49   // Inicializar SPIFFS
50   if (!SPIFFS.begin(true)) {
51     Serial.println("Error al montar SPIFFS");
52     return;
53   }
```

Con el SPIFFS inicializado se procede a la apertura de los archivos de rotación y a su volcado en los matices `matrixTR` paso de rotación, `matrixROTRP` paso de rotación para salir de reposo y `matrixROTRN` paso de rotación para retornar a reposo. Para esta operación se emplea la función `parseline` que se explicará en el siguiente apartado.

Para finalizar el void set up se inicializan los módulos PCA9685:

```
102   pwm.begin();
103   pwm.setPWMFreq(50);
104   delay(10);
105   pwm1.begin();
106   pwm1.setPWMFreq(50);
107 }
```

5.4.3 Declaración de funciones

5.4.3.1 Función para introducir datos en matices (`parseline`).

Se trata de una función que toma como parámetros la matriz de destino, el número de columnas y filas de la matriz, y una fila completa del archivo .txt. La función separa los valores basándose en los espacios entre ellos y luego los guarda en las matrices que almacenan las trayectorias de las patas.

```
1  //Funcion para introducir los datos leídos a las matices de
2  trayectorias
3  void parseLine(String line, int row, uint16_t* matrix, int cols) {
4    int col = 0;
5    int start = 0;
6    for (int i = 0; i < line.length(); i++) {
7      if (line[i] == ' ') {
8        // Si encontramos un espacio, extraemos el valor entre
9  'start' e 'i'
10       String valueStr = line.substring(start, i);
11       int valueint = valueStr.toInt();
12       // Asignamos el valor convertido a entero a la matriz
13       matrix[row * cols + col] = (uint16_t)valueint;
14       col++;
15       start = i + 1;
16     }
17   }
18
19   // Agregar el último valor de la línea
```

```
20    if (start < line.length()) {
21      // Extraemos el último valor después del último espacio
22      String valueStr = line.substring(start);
23      int valueint = valueStr.toInt();
24      matrix[row * cols + col] = (uint16_t)valueint;
      }
    }
```

5.4.3.2 Modificar modo de caminar (`cambiamodo`).

Esta función recibe el argumento, modo y dependiendo del modo elegido carga en las matrices de avance lineal: `matrixFW, matrixRO, matrixRN` el modo de caminar seleccionado.

```
10 //Funcion para modificar el modo de caminar del hexapod
11 void cambiomodo(char modo){
12 //Cargamos el nombre del archivo a leer dependiendo del modo
13 if (modo == 'N') {
14   strcpy(fileName1, "/forward.txt");//Archivo 1
15   strcpy(fileName2, "/reposo.txt");// Archivo 2
16   strcpy(fileName3, "/return.txt");// Archivo 3
17   }
18 if (modo == 'U') {
19   strcpy(fileName1, "/forwardup.txt");// Archivo 1
20   strcpy(fileName2, "/reposoup.txt");// Archivo 2
21   strcpy(fileName3, "/returnup.txt");// Archivo 3
22   }
23 if (modo == 'O') {
24   strcpy(fileName1, "/forwardobs.txt");// Archivo 1
25   strcpy(fileName2, "/reposoobs.txt");// Archivo 2
26   strcpy(fileName3, "/returnobs.txt");// Archivo 3
27   }
28   // Abrir el archivo 1
29   File fileF = SPIFFS.open(fileName1, "r");
30   if (!fileF) {
31     Serial.println("Error al abrir el archivo 1");
32     return;
33   }
34   // Leer el archivo 1
35   int row = 0;
36   while (fileF.available() && row < rows) {
37     String line = fileF.readStringUntil('\n');
38     parseLine(line, row, &matrixFW[0][0], cols);
39     row++;
40   }
41   fileF.close();
42
43   // Abrir el archivo 2
44   File fileR = SPIFFS.open(fileName2, "r");
45   if (!fileR) {
46     Serial.println("Error al abrir el archivo 2");
47     return;
48   }
49   // Leer el archivo 2
50   row = 0;
51   while (fileR.available() && row < rowsA) {
52     String line = fileR.readStringUntil('\n');
```

```
53      parseLine(line, row, &matrixRP[0][0], cols);//, tiemposFP);
54      row++;
55    }
56    fileR.close();
57
58    // Abrir el archivo 3
59    Serial.println(fileName3);
60    File fileN = SPIFFS.open(fileName3, "r");
61    if (!fileN) {
62      Serial.println("Error al abrir el archivo 3");
63      return;
64    }
65    // Leer el archivo 3
66    row = 0;
67    while (fileN.available() && row < rowsA) {
68      String line = fileN.readStringUntil('\n');
69      parseLine(line, row, &matrixRN[0][0], cols);//, tiemposFP);
70      row++;
71    }
72    fileN.close();
73  }
```

5.4.3.3 Función para enviar ticks(`enviarticks`).

Función encargada del envío de los ticks a los diferentes canales del módulo PCA9685, recibe como argumento los 18 ticks a enviar. El módulo 0 presenta un desfase de -17 ticks que fue medido en laboratorio mediante un osciloscopio, por ello se suma +17 ticks en los canales de ese módulo.

```
1  //funcion para enviar los ticks a los servomotores
2  void enviaticks(int P1ticks1, int P1ticks2, int P1ticks3, int
3  P2ticks1, int P2ticks2, int P2ticks3, int P3ticks1, int P3ticks2,
4  int P3ticks3, int P4ticks1, int P4ticks2, int P4ticks3, int
5  P5ticks1, int P5ticks2, int P5ticks3, int P6ticks1, int P6ticks2,
6  int P6ticks3) {
7      //PATA 1
8      pwm.setPWM(1, 0, P1ticks1 + 17);
9      pwm.setPWM(2, 0, P1ticks2 + 17);
10     pwm.setPWM(3, 0, P1ticks3 + 17);
11     //PATA 2
12     pwm.setPWM(4, 0, P2ticks1 + 17);
13     pwm.setPWM(5, 0, P2ticks2 + 17);
14     pwm.setPWM(6, 0, P2ticks3 + 17);
15     //PATA 3
16     pwm.setPWM(7, 0, P3ticks1 + 17);
17     pwm.setPWM(8, 0, P3ticks2 + 17);
18     pwm.setPWM(9, 0, P3ticks3 + 17);
19     //PATA 4
20     pwm1.setPWM(1, 0, P4ticks1);
21     pwm1.setPWM(2, 0, P4ticks2);
22     pwm1.setPWM(3, 0, P4ticks3);
23     //PATA 5
24     pwm1.setPWM(4, 0, P5ticks1);
```

```
25      pwm1.setPWM(5, 0, P5ticks2);
26      pwm1.setPWM(6, 0, P5ticks3);
27      //PATA 6
28      pwm1.setPWM(7, 0, P6ticks1);
29      pwm1.setPWM(8, 0, P6ticks2);
30      pwm1.setPWM(9, 0, P6ticks3);
31  }
```

5.4.3.4 Void loop.

Finalmente se accede al void loop que se ejecutara de forma cíclica siguiendo los siguientes pasos:

1. Limpiar el buffer del Bluetooth.
2. Esperar y leer el dato recibido por BT, el dato es un carácter (variable tipo char).
3. Si el carácter recibido es una N, U o una O se llama a la función `cambiamodo` con el dato como argumento.
4. En caso contrario se compara el carácter recibido que puede ser A, F, D, R o L para ejecutar la acción de, volver a reposo A, caminar hacia delante F, hacia detrás D, giro derecha R o giro izquierda L.
5. Si al entrar en un modo de movimiento: F, D ,R o L desde un estado de reposo se ejecuta la acción paso desde reposo almacenada en la matriz de trayectoria correspondiente.
6. Guardar el valor de la variable en `previo` para conocer el estado desde que parten los movimientos del hexápodo.
7. Repetir.

Casos girar izquierda L y marcha atrás D:

Las matrices de trayectorias almacenadas están diseñadas para ser utilizadas en desplazamientos hacia adelante en el caso de `matrixFW` y en giros en sentido horario en el caso de `matrixROT`. Para lograr un movimiento contrario al programado, se leen las matrices de forma inversa. Esta lectura inversa de las matrices se aplica tanto para el paso de salida de reposo como para el paso de retorno, pero ahora utilizaremos el paso de retorno para salir de reposo y el paso de reposo para volver a reposo.

Ejemplo del caso:

```
1   //-----------------------GIRO LEFT-----------------------------
2   --
3     if (dato == 'L') {
4   //Si partimos desde reposo ejecutamos el giro para salir de
5   reposo
6       if (previo == 'A') {
7         for (int i = rowsA - 1; i >= 0; i--) {
8           // Asignación de valores a variables específicas
9           P1ticks1 = matrixROTRN[i][0];
10          P1ticks2 = matrixROTRN[i][1];
11          P1ticks3 = matrixROTRN[i][2];
12          P2ticks1 = matrixROTRN[i][3];
```

```
13          P2ticks2 = matrixROTRN[i][4];
14          P2ticks3 = matrixROTRN[i][5];
15          P3ticks1 = matrixROTRN[i][6];
16          P3ticks2 = matrixROTRN[i][7];
17          P3ticks3 = matrixROTRN[i][8];
18          P4ticks1 = matrixROTRN[i][9];
19          P4ticks2 = matrixROTRN[i][10];
20          P4ticks3 = matrixROTRN[i][11];
21          P5ticks1 = matrixROTRN[i][12];
22          P5ticks2 = matrixROTRN[i][13];
23          P5ticks3 = matrixROTRN[i][14];
24          P6ticks1 = matrixROTRN[i][15];
25          P6ticks2 = matrixROTRN[i][16];
26          P6ticks3 = matrixROTRN[i][17];
27
28        enviaticks( P1ticks1,  P1ticks2,  P1ticks3,  P2ticks1,
29  P2ticks2,  P2ticks3,  P3ticks1,  P3ticks2,  P3ticks3,  P4ticks1,
30  P4ticks2,  P4ticks3,  P5ticks1,  P5ticks2,  P5ticks3,  P6ticks1,
31  P6ticks2,  P6ticks3);
32        }
33      }
34      previo = dato;
35      for (int i = rows; i >= 0; i--) {
36        // Asignación de valores a variables específicas
37        P1ticks1 = matrixRT[i][0];
38        P1ticks2 = matrixRT[i][1];
39        P1ticks3 = matrixRT[i][2];
40        P2ticks1 = matrixRT[i][3];
41        P2ticks2 = matrixRT[i][4];
42        P2ticks3 = matrixRT[i][5];
43        P3ticks1 = matrixRT[i][6];
44        P3ticks2 = matrixRT[i][7];
45        P3ticks3 = matrixRT[i][8];
46        P4ticks1 = matrixRT[i][9];
47        P4ticks2 = matrixRT[i][10];
48        P4ticks3 = matrixRT[i][11];
49        P5ticks1 = matrixRT[i][12];
50        P5ticks2 = matrixRT[i][13];
51        P5ticks3 = matrixRT[i][14];
52        P6ticks1 = matrixRT[i][15];
53        P6ticks2 = matrixRT[i][16];
55        P6ticks3 = matrixRT[i][17];
56
57      enviaticks( P1ticks1,  P1ticks2,  P1ticks3,  P2ticks1,
58  P2ticks2,  P2ticks3,  P3ticks1,  P3ticks2,  P3ticks3,  P4ticks1,
59  P4ticks2,  P4ticks3,  P5ticks1,  P5ticks2,  P5ticks3,  P6ticks1,
60  P6ticks2,  P6ticks3);
61      }
62    }
```

5.5 Desarrollo de la aplicación móvil en App Inventor

Para el envío de los comandos vía BT al ESP32 se desarrolló una App para móvil, la interfaz de esta y su modo de empleo puede ser consultado en el Anexo 10.6. En el anexo se adjunta también el enlace para la descarga de la App y de su código completo.

La funcionalidad de la App es sencilla. Primeramente, se ha de vincular el Bluetooth de ambos dispositivos una vez vinculado se envían caracteres vía BT al ESP32 dependiendo del botón que se mantenga pulsado o una "A" si es que no se pulsa nada.

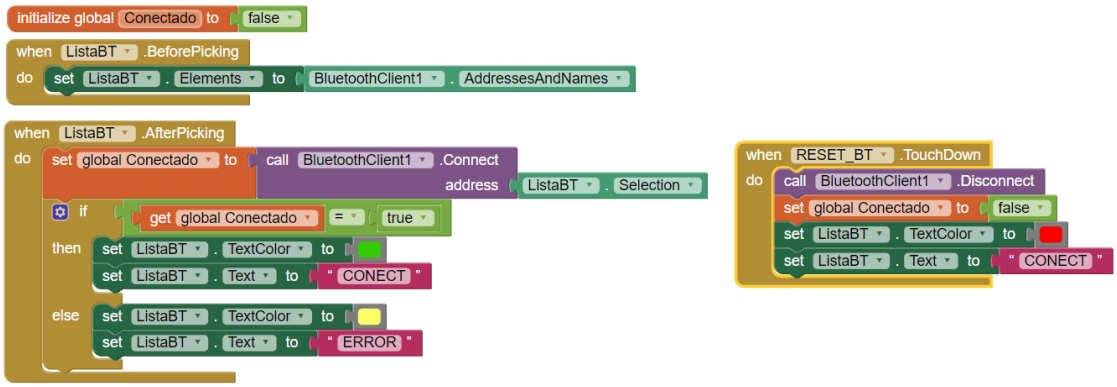


Figura 114. Parte del código que administrar la conexión y desconexión del BT. Fuente: Elaboración propia.

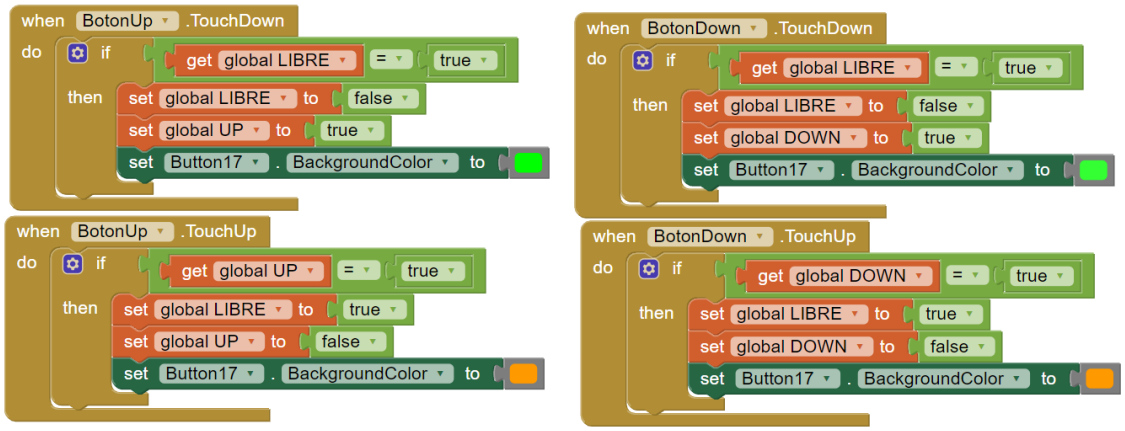


Figura 115. Código que registra la pulsación del BotonUp y el BotonDown para que este envié un carácter mientras es pulsado. Fuente: Elaboración propia.

En la Figura 115 se puede apreciar también como se protege de pulsaciones simultáneas mediante la variable `LIBRE` que indica que no hay botones presionados.

Por último, se emplea un `clock` para comprobar el estado de las variables a intervalos constantes y enviar el valor de la letra correspondiente a la variable activa.

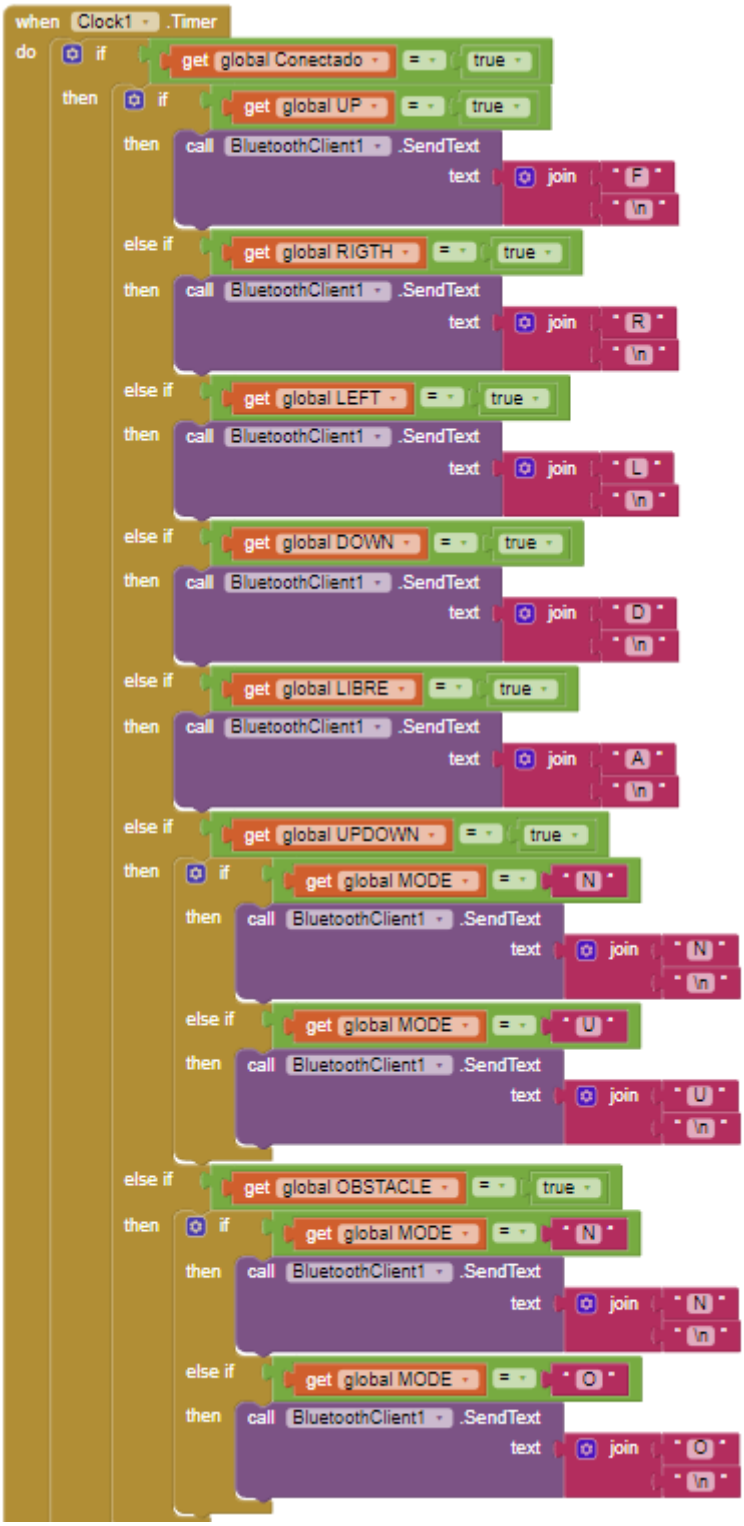


Figura 116. Comprobación del estado de las variables y envío del carácter vía BT. Fuente: Elaboración propia.

6 Resultados

En este capítulo se presentarán los resultados obtenidos durante el desarrollo y la prueba del robot hexápodo. Se describirán las pruebas de funcionamiento realizadas, se analizarán los resultados obtenidos, y se discutirán las limitaciones encontradas junto con posibles mejoras para el futuro.

6.1 Pruebas de funcionamiento

Para evaluar el desempeño del robot hexápodo, se realizaron diversas pruebas de funcionamiento. Estas pruebas fueron grabadas y se encuentran disponibles en los siguientes enlaces:

- Vídeo 1: Control del robot mediante una aplicación móvil. En este vídeo se muestra al hexápodo posado sobre una superficie elevada sin que las patas toquen el piso. Se observa cómo se controla el robot a través de la aplicación móvil, demostrando la capacidad de enviar comandos y recibir respuestas adecuadas.
- Vídeo 2: Movimientos de rotación y traslación. Este vídeo muestra al hexápodo sobre el suelo, realizando movimientos de traslación hacia adelante y hacia atrás, así como rotaciones en sentido horario y antihorario.
- Vídeo 3: Rookie en entorno "hostil". En este vídeo se muestra al robot hexápodo en un entorno más irregular superando una pequeña rampa descendente con el paso alto y saliendo de la base.
- Vídeo 4: Movimientos al completo. Este vídeo muestra todos los movimientos implementados en nuestro robot. Caminar hacia delante/atrás, rotar sentido horario/antihorario, paso normal/elevado y el cambio de altura en z del robot.

El objetivo de estas pruebas y vídeos es demostrar la funcionalidad básica del robot, la precisión de sus movimientos y la capacidad de controlarlo de manera remota mediante una aplicación móvil.

6.2 Análisis de resultados

Los resultados obtenidos en las pruebas de funcionamiento del robot hexápodo han sido muy satisfactorios. En primer lugar, se destaca que el movimiento del robot es notablemente fluido, lo cual indica una correcta sincronización e implementación de los algoritmos de control además de un hardware correctamente elegido. Además, la respuesta del robot a los comandos enviados desde la aplicación móvil es rápida y precisa, mostrando una integración eficaz entre el hardware y el software de control.

El ensamblaje del robot ha demostrado ser robusto, permitiendo que el hexápodo se mantenga en pie de manera estable durante las pruebas. Asimismo, todas las patas del robot ejecutan los movimientos de locomoción de manera coordinada y correcta, lo que refleja una calibración precisa de los motores.

El control del robot a través de la aplicación móvil es simple e intuitivo, facilitando su operación incluso para usuarios sin experiencia previa.

6.3 Limitaciones y mejoras

A pesar de los resultados positivos obtenidos, se han identificado algunas limitaciones y áreas de mejora para el robot hexápodo. Una de las principales limitaciones es la incapacidad del robot para adaptarse a entornos nuevos o irregulares, ya que los movimientos actuales están prefijados y no permiten una adaptación dinámica a diferentes tipos de terreno. Para mejorar esta situación, se podría implementar un sistema de sensores que permita la adaptación del robot a diversos entornos.

Otra área de mejora es la capacidad de movimiento omnidireccional. Actualmente, el robot no puede avanzar de manera diagonal sin tener que rotar primero. Desarrollar un sistema de control que permita movimientos omnidireccionales aumentaría significativamente la versatilidad del robot.

Además, la integración de sensores adicionales como IMUs, cámaras y sensores de proximidad podría mejorar considerablemente la capacidad del robot para navegar en entornos complejos. Sensores de este tipo podrían proporcionarían información en tiempo real, permitiendo ajustar los movimientos del robot y evitar obstáculos.

Finalmente, la aplicación móvil podría mejorarse para incluir funcionalidades adicionales, como la capacidad de programar secuencias de movimientos, ajustar parámetros de movimiento en tiempo real y proporcionar retroalimentación visual del estado del robot. Estas mejoras aumentarían la funcionalidad del robot y también abrirían nuevas posibilidades para su aplicación en diversos campos, desde la investigación hasta la educación y el entretenimiento.

7 Presupuesto

El presupuesto se divide en dos partidas fundamentales: Partida de materiales y partida de mano de obra.

La partida de materiales se separa en las siguientes unidades de obra: Tornillería y piezas 3D, Hardware electrónico y Software y licencias.

Cálculo del Costo de la Impresión 3D:

Para estimar el costo de la impresión 3D de las piezas del robot hexápodo, consideramos dos factores principales: el costo del material de impresión (filamento PLA) y el costo de la electricidad consumida por la impresora durante el proceso de impresión. Para estos cálculos, se asumirá que todas las piezas se imprimieron utilizando una impresora Prusa MK3.

Costo del Material de Impresión:

El costo del filamento PLA puede variar, pero en promedio, un kilogramo de filamento PLA cuesta alrededor de 20 euros. Usaremos el software Cura para obtener el costo del material por cada pieza.

Aquí está una lista estimada de piezas y sus costos de material basados en el software Cura:

Pieza	*Peso (g)*	*Costo material (€)*
Caja Servo	15	0,31
U 68	12	0,24
U 71.4	12	0,24
Fémur (parte inf.)	18	0,36
Fémur (parte sup.)	5	0,09
Tibia	25	0,5
Cuerpo (parte inf.)	45	0,89
Cuerpo (parte sup.)	96	1,93

Tabla 8. Piezas impresas en 3D con su respectivo peso en gramos y el costo de material PLA en euros (parámetros extraídos del software UltiMaker Cura)

La impresora Prusa MK3 tiene un consumo de aproximadamente 120W durante la impresión. El costo promedio de la electricidad en España es de aproximadamente 0.20 euros por kWh [64] [65].

Para calcular el costo de electricidad, utilizamos la siguiente fórmula:

$$\text{Costo electricidad} = \text{Consumo}(kW) \times \text{Tiempo de impresión}(h) \times \textit{Precio electricidad}(€/kWh)$$

Aquí se presenta una tabla con el tiempo estimado de impresión de cada pieza y el costo correspondiente de electricidad:

Pieza	*Tiempo de impresión (h)*	*Coste electricidad (€)*
Caja Servo	1,72	0,04
U 68	1,37	0,03
U 71.4	1,35	0,03
Fémur (parte inf.)	2,1	0,05
Fémur (parte sup.)	0,53	0,01
Tibia	2,85	0,07
Cuerpo (parte inf.)	4,95	0,12
Cuerpo (parte sup.)	10,87	0,26

Tabla 9. Tabla de Costos de Electricidad para la Impresión 3D (parámetros de tiempo extraídos del software UltiMaker Cura)

Sumando el costo del material y el costo de la electricidad, obtenemos el costo total de impresión para cada pieza:

Pieza	*Costo material (€)*	*Coste electricidad (€)*	*Costo total (€)*
Caja Servo	0,31	0,04	0,35
U 68	0,24	0,03	0,27
U 71.4	0,24	0,03	0,27
Fémur (parte inf.)	0,36	0,05	0,41
Fémur (parte sup.)	0,09	0,01	0,1
Tibia	0,5	0,07	0,57
Cuerpo (parte inf.)	0,89	0,12	1,01
Cuerpo (parte sup.)	1,93	0,26	2,19

Tabla 10. Tabla de Costos Totales de Impresión 3D

Estos cálculos permiten tener una visión detallada del costo de impresión 3D de cada pieza del robot hexápodo, facilitando así una estimación precisa para el presupuesto del proyecto.

Continuaremos detallando los costos de los componentes electrónicos y licencias, así como la mano de obra en las siguientes secciones.

Separada en sus distintas unidades de obra, el coste de la partida de materiales es el siguiente:

ID	Pieza	Descripción	Ctd.	P. Unit.	Total
1	Caja Servo	Soporte del servomotor para el codo	6	0,35 €	2,10 €
2	U 68	Conector en U para unir codo y fémur, 68mm	6	0,27 €	1,62 €
3	U 71.4	Conector en U para unir fémur y tibia, 71.4mm	12	0,27 €	3,24 €
4	Fémur (parte inf.)	Soporte inferior de servomotores en el fémur	6	0,41 €	2,46 €
5	Fémur (parte sup.)	Tapa superior de servomotores en el fémur	6	0,10 €	0,60 €
6	Tibia	Parte final de la pata del robot	6	0,57 €	3,42 €
7	Cuerpo (parte inf.)	Base inferior del robot hexápodo	1	1,01 €	1,01 €
8	Cuerpo (parte sup.)	Tapa superior del robot hexápodo	1	2,19 €	2,19 €
9	Tornillo M2 8mm	Tornillo pequeño para componentes ligeros	72	0,32 €	23,04 €
10	Tornillo M3 15mm	Tornillo mediano para caja del servo	10	0,18 €	1,75 €
11	Tornillo M3 20mm	Tornillo para piezas como las U	36	0,17 €	6,14 €
12	Tornillo M3 40mm	Tornillo largo para el cuerpo del robot	12	0,27 €	3,29 €
13	Tornillo M3 50mm	Tornillo largo para fémur	24	0,70 €	16,82 €
14	Arandela M3	Distribuye la carga de tornillos M3	120	0,06 €	7,50 €
15	Tuerca M3	Asegura tornillos M3	90	0,03 €	3,11 €
		Subtotal Tornillería y Piezas 3D			***78,29 €***

Tabla 11. Partida de materiales. Unidad de obra: Tornillería y piezas 3D

ID	Pieza	Descripción	Ctd.	P. Unit.	Total
16	Servo DS-S020A-C	Servomotor digital de 30 kg/cm para coxa y tibia	12	22,22 €	266,64 €
17	Servo SPT5435LV	Servomotor digital de 35 kg/cm para fémur	6	29,19 €	175,14 €
18	ESP32	Microcontrolador principal del robot	1	17,75 €	17,75 €
19	PCA9685	Módulo controlador PWM de 16 canales	2	22,80 €	45,60 €
20	Pololu U1V11A	Regulador de tensión de 5V para la ESP32	1	17,55 €	17,55 €
21	Batería	Batería Li-ion SAMSUNG INR21700-40T	2	4,75 €	9,50 €
22	Cables 20AWG	Cables para señales PWM	1	8,49 €	8,49 €
23	Surtido de punteras	Punteras para conexiones de cables	1	11,99 €	11,99 €
24	Regletas electrónicas	Conectores para distribuir alimentación	1	8,99 €	8,99 €
25	Manguera espilaradas	Material para organizar cables	1	3,23 €	3,23 €
26	Termo retráctil	Tubo para aislar conexiones soldadas	1	0,60 €	0,60 €
27	Tira de pines	Pines para conexiones entre módulos y servos	2	2,30 €	4,60 €
28	Cable paralelo 2x0.5	Cable de alimentación de 0.5 mm^2	1	0,40 €	0,40 €
29	Cable paralelo 2x0.75	Cable de alimentación de 0.75 mm^2	1	0,60 €	0,60 €
30	Cable paralelo 2x1.5	Cable de alimentación de 1.5 mm^2	1	0,75 €	0,75 €
31	Conectores XT-60	Conectores para la batería	1	4,1 €	4,10 €
		Subtotal Hardware electrónico			***575,93 €***

Tabla 12. Partida materiales. Unidad de obra: Hardware electrónico

ID	Programa	Descripción	Ctd.	P. Unit.	Total
32	Octave	Software de cálculo numérico	1	0,00 €	0,00 €
33	Fusion 360	Software CAD para diseño 3D del robot	1	506,85 €	506,85 €
34	Arduino IDE	Entorno para programar el ESP32	1	0,00 €	0,00 €
35	Visual Studio Code	Editor de código para desarrollar scripts	1	0,00 €	0,00 €
36	Microsoft Office	Suite ofimática para redacción de documentos	1	92,99 €	92,99 €
37	KiCad	Software para diseño del esquemático	1	0,00 €	0,00 €
		Subtotal Software y licencias			***599,84 €***

Tabla 13. Partida de materiales. Unidad de obra: Software y licencias

Unidad de obra	Ctd.	P. Unit.	Total
Tornillería y piezas 3D	1	78,29 €	78,29 €
Hardware electrónico	1	575,93 €	575,93 €
Software y licencias	1	599,84 €	599,84 €
	Subtotal material		***1.254,06 €***

Tabla 14. Partida de materiales. Subtotal

Tomando en cuenta 25 €/hora de mano de obra del ingeniero, 25 €/hora de estudio y verificación y 25 €/h de redacción, el coste de mano de obra desglosado asciende a:

Unidad de obra	Horas	P. Hora	Total
Estudio previo	40	25,00 €	1.000,00 €
Diseño hardware	180	25,00 €	4.500,00 €
Codificación software	260	25,00 €	6.500,00 €
Verificación	40	25,00 €	1.000,00 €
Informe	80	25,00 €	2.000,00 €
	Subtotal		***15.000,00 €***

Tabla 15. Partida de mano de obra.

La suma de las distintas partidas será:

Partida		***Total***
Material		1.254,06 €
Mano de obra		15.000,00 €
	Subtotal	16.254,06 €
	IGIC (7%)	1.137,78 €
	Total	***17.391,84 €***

Tabla 16. Coste total del presupuesto

El total de proyecto, incluyendo impuestos, asciende a diecisiete mil trescientos noventa y un euros con ochenta y cuatro céntimos (17.391,84 €).

8 Conclusiones

Como resultado del desarrollo y la experimentación realizados en este trabajo, se han obtenido varias conclusiones importantes que subrayan el éxito del proyecto en distintos aspectos. Se ha demostrado que es posible diseñar y construir un robot hexápodo utilizando tecnología de impresión 3D y componentes electrónicos accesibles. La estructura mecánica, diseñada en el software Fusion 360 y fabricada con filamento PLA, ha mostrado ser lo suficientemente robusta para soportar todas las operaciones necesarias del robot.

Además, la elección del microcontrolador ESP32 ha sido correcta debido a su versatilidad y potencia. Este microcontrolador ha mostrado una excelente capacidad para manejar múltiples servomotores. Su compatibilidad con el entorno de desarrollo Arduino IDE ha facilitado la programación y reutilización de códigos previos, optimizando así el proceso de desarrollo.

El uso de módulos PWM (PCA9685) ha simplificado enormemente el control de los servomotores, permitiendo una gestión eficiente de las señales PWM necesarias. La comunicación mediante el protocolo I2C ha demostrado ser confiable y efectiva para la transmisión de comandos de control desde el ESP32 hacia los motores. Las pruebas realizadas han confirmado que el robot hexápodo es capaz de ejecutar movimientos complejos y mantener la estabilidad. La respuesta del robot a los comandos enviados desde la aplicación móvil ha sido rápida y precisa, lo que demuestra la efectividad del sistema de control implementado.

Finalmente, la modificación del sistema de cableado, cambiando de conexiones con cables DuPont a conexiones soldadas, ha mejorado significativamente la fiabilidad del robot. Esta mejora ha reducido considerablemente los problemas de desconexiones e interferencias que ocasionaban movimientos indeseados de las patas, asegurando un correcto funcionamiento. En resumen, este trabajo ha permitido no solo la creación de un robot hexápodo funcional, sino también la validación de técnicas y componentes accesibles que pueden ser utilizados en futuros desarrollos robóticos.

CONCLUSIONS

As a result of the development and experimentation carried out in this work, several important conclusions have been reached, highlighting the success of the project in various aspects. It has been demonstrated that it is possible to design and construct an efficient hexapod robot using 3D printing technology and accessible electronic components. The mechanical structure, designed in Fusion 360 software and fabricated with PLA filament, has proven to be robust enough to support all necessary operations of the robot.

Furthermore, the choice of the ESP32 microcontroller has been correct due to its versatility and power. This microcontroller has shown excellent capability in handling multiple servomotors. Its compatibility with the Arduino IDE development environment has facilitated programming and the reuse of previous codes, thus optimizing the development process.

The use of PWM modules (PCA9685) has simplified the control of the servomotors, allowing for efficient management of the necessary PWM signals. Communication via the I2C protocol has proven to be reliable and effective for transmitting control commands from the ESP32 to the motors. The tests conducted have confirmed that the hexapod robot is capable of executing complex movements and maintaining stability. The robot's response to commands sent from the mobile application has been quick and precise, demonstrating the effectiveness of the implemented control system.

Finally, the modification of the wiring system, changing from DuPont cable connections to soldered connections, has significantly improved the robot's stability and reliability. This improvement has reduced the issues of disconnections and undesired movements of the legs, ensuring more consistent and secure operation. In summary, this work has not only enabled the creation of a functional hexapod robot but also validated accessible techniques and components that can be used in future robotic developments.

9 Referencias bibliográficas

[1] G. Paralea, «HACKADAY.IO,» 17 Febrero 2021. [En línea]. Available: https://hackaday.io/project/177750/logs?sort=oldest. [Último acceso: 25 Junio 2024].

[2] A. Moreno, «MORFOLOGIA EXTERNA DE INSECTOS,» La PLata.

[3] «Robotics Tomorrow,» 8 Septiembre 2023. [En línea]. Available: https://www.roboticstomorrow.com/news/2023/08/09/new-hexapod-applications-for-precision-motion-control-from-pi/20934/. [Último acceso: 25 Junio 2024].

[4] D. Collins, «LinearMotion,» [En línea]. Available: https://www.linearmotiontips.com/hexapod-robot-design-variations-applications/. [Último acceso: 25 Junio 2024].

[5] A. Beyatli, «Acrome,» [En línea]. Available: https://acrome.net/post/what-are-hexapod-robots-used-for. [Último acceso: 25 Junio 2024].

[6] D. Industry.

[7] «Robots Guide,» 2001. [En línea]. Available: https://robotsguide.com/robots/rhex. [Último acceso: 3 Julio 2024].

[8] «Robots Guide,» 2013. [En línea]. Available: https://robotsguide.com/robots/lauron?video=4. [Último acceso: 3 Julio 2024].

[9] «Robots Guide,» 2009. [En línea]. Available: https://robotsguide.com/robots/dash?video=2. [Último acceso: 3 Julio 2024].

[10] «Robots Guide,» 1989. [En línea]. Available: https://robotsguide.com/robots/genghis. [Último acceso: 3 Julio 2024].

[11] «Robots Guide,» 2017. [En línea]. Available: https://robotsguide.com/robots/daisy. [Último acceso: 3 Julio 2024].

[12] S. Briot, «frontiers,» 8 Septiembre 2023. [En línea]. Available: https://www.frontiersin.org/journals/robotics-and-ai/articles/10.3389/

frobt.2023.1282798/full. [Último acceso: 3 Julio 2024].

[13] V. Abarca, «MDPI,» 20 Septiembre 2023. [En línea]. Available: https://www.mdpi.com/2218-6581/12/5/131. [Último acceso: 3 Julio 2024].

[14] G. Torres, «Urany,» 13 Abril 2022. [En línea]. Available: https://urany.net/blog/robots-delta-qu%C3%A9-c%C3%B3mo-cu%C3%A1ndo-y-por-qu%C3%A9. [Último acceso: 3 Julio 2024].

[15] D. Collins, «Linear Motion Tips,» [En línea]. Available: https://www.linearmotiontips.com/what-are-hexapod-robots-stewart-platforms/. [Último acceso: 3 Julio 2024].

[16] J.-P.Merlet, Parallel Robots, Springer Science & Business Media, 2006.

[17] O. Gonzalez, «BricoGeek,» 11 Marzo 2008. [En línea]. Available: https://blog.bricogeek.com/noticias/diy/video-hexapod-phoenix-el-robot-arana/. [Último acceso: 3 Julio 2024].

[18] D. Burón, «ITespresso,» 10 Diciembre 2012. [En línea]. Available: https://www.itespresso.es/quereis-robots-cuadrupedos-y-hexapodos-aqui-teneis-de-los-mas-avanzados-56352.html. [Último acceso: 3 Julio 2024].

[19] «Borunte,» 31 Octubre 2022. [En línea]. Available: http://es.borunte.net/info/difference-of-serial-robot-and-parallel-robot-76625982.html.
[Último acceso: 3 Julio 2024].

[20] C. S. Gurel, «hackaday.io,» 29 Junio 2017. [En línea]. Available: https://hackaday.io/project/21904-hexapod-modelling-path-planning-and-control/log/62326-3-fundamentals-of-hexapod-robot. [Último acceso: 4 Julio 2024].

[21] A. R. X. D. Zhiying Wang, «Analysis of Typical Locomotion of a Symmetric Hexapod Robot,» *Robotica,* vol. 28, nº 6, pp. 893-907, 2010.

[22] R. T. R. C. T. A. R. B. A. J. I. &. D. F. Pavan Ramdya, «Climbing favours the tripod gait over alternative faster insect gaits,» *Nature Communications,* vol. 8, nº 14494, 2017.

[23] N. H. Darbha, «CORE Scholar,» 12 Diciembre 2017. [En línea]. Available:

https://corescholar.libraries.wright.edu/cgi/viewcontent.cgi?article=3006&context=etd_all. [Último acceso: 4 Julio 2024].

[24] tyberius, «Community RobotShop,» 10 Agosto 2010. [En línea]. Available: https://community.robotshop.com/robots/show/interbotix-phantomx-hexapod. [Último acceso: 25 Junio 2024].

[25] Segway, «Segway,» [En línea]. Available: https://es-es.segway.com/. [Último acceso: 2024 Junio 2024].

[26] N. Correl, «LibreTexts,» [En línea]. Available: https://espanol.libretexts.org/Ingenieria/Ingenier%C3%ADa_Mec%C3%A1nica/Introducci%C3%B3n_a_los_Robots_Aut%C3%B3nomos_(Correll)/02%3A_Locomoci%C3%B3n_y_Manipulaci%C3%B3n/2.02%3A_Estabilidad_est%C3%A1tica_y_din%C3%A1mica. [Último acceso: 25 Junio 2024].

[27] E. Rodríguez, «Canal Innova,» [En línea]. Available: https://canalinnova.com/como-se-aplica-la-robotica-en-la-educacion-ventajas-e-inconvenientes/. [Último acceso: 25 Junio 2024].

[28] D. Wise, «GeaLab,» [En línea]. Available: https://www.techgearlab.com/reviews/cool-gadgets/3d-printer/creality-3d-cr-10s. [Último acceso: 25 Junio 2024].

[29] J. D., «3Dnatives,» 12 Octubre 2017. [En línea]. Available: https://www.3dnatives.com/en/testing-witbox-2-3d-printer/. [Último acceso: 25 Junio 2024].

[30] G. Dunham, «markershop,» 7 Junio 2022. [En línea]. Available: https://makershop.co/prusa-i3-mk3s-review/. [Último acceso: 25 Junio 2024].

[31] All3DP, «All3DP,» 17 Febreo 2016. [En línea]. Available: https://all3dp.com/bq-hephestos-2-review-3d-printer-depth-test/. [Último acceso: 25 Junio 2024].

[32] «BricoGeek,» [En línea]. Available: https://tienda.bricogeek.com/arduino-compatibles/

1274-nodemcu-esp32-wroom-wifi-bluetooth.html. [Último acceso: 25 Junio 2024].

[33] Kuongshun. [En línea]. Available: https://www.szks-kuongshun.com/uno/uno-

board-shield/mega-sensor-shield-v2-0-for-mega-2560-r3.html.

[34] «tiendatec,» [En línea]. Available: https://www.tiendatec.es/maker-zone/microcontroladores/420-arduino-mega-2560-r3-atmega2560-compatible-cable-usb-b-gratis-8404201160013.html.

[35] M. Akbari, «electropeak,» [En línea]. Available: https://electropeak.com/learn/raspberry-pi-serial-communication-uart-w-arduino-pc/. [Último acceso: 25 Junio 2024].

[36] «Turibot,» [En línea]. Available: https://www.turibot.es/modulo-bluetooth-hc-05. [Último acceso: 25 Junio 2024].

[37] H. Tailor, «Markerguides,» 29 Enero 2024. [En línea]. Available: https://www.makerguides.com/esp32-vs-arduino-speed-comparison/. [Último acceso: 25 Junio 2024].

[38] R. Hernandez, «Enmbedwiz,» 30 Marzo 2023. [En línea]. Available: https://embedwiz.com/arduino-vs-esp32/. [Último acceso: 25 Junio 2024].

[39] «simplyiotsensors,» 27 Mayo 2023. [En línea]. Available: https://www.simplyiotsensors.com/2023/05/Arduino-vs-ESP-32.html. [Último acceso: 25 Junio 2024].

[40] aprendiendoarduino. [En línea]. Available: https://aprendiendoarduino.wordpress.com/tag/motor-paso-a-paso/. [Último acceso: 25 Junio 2024].

[41] HobbyKing. [En línea]. Available: https://hobbyking.com/es_es/towerpro-mg996r-360-robotic-servo-11kg-0-015sec-55g.html. [Último acceso: 25 Junio 2024].

[42] «Ubuy,» [En línea]. Available: https://www.ubuy.co.in/product/FMA65YJ8G-dspower-30kg-waterproof-digital-servo-metal-gear-high-torque-motor-servos-with-25t-servo-horn-for-1-8-1-10-1-12-rc-crawler-car-baja-robot-boat-helicop. [Último acceso: 25 Junio 2024].

[43] «Amazon,» [En línea]. Available: https://www.amazon.es/Dilwe-Engranaje-Digital-SPT5435LV-Compatible/dp/B099KNV5SK. [Último acceso: 25 Junio 2024].

[44] «Instructables,» [En línea]. Available: https://www.instructables.com/Mastering-

Servo-Control-With-PCA9685-and-Arduino/. [Último acceso: 25 Junio 2024].

[45] D. Mittal, «Circuit Digest,» 12 Febrero 2024. [En línea]. Available: https://circuitdigest.com/microcontroller-projects/pca9685-multiple-servo-control-using-arduino. [Último acceso: 25 Junio 2024].

[46] «Bettery Junction,» [En línea]. Available: https://www.batteryjunction.com/samsung-40t-21700-battery. [Último acceso: 26 Junio 2024].

[47] Eric, «Esaler Electronic,» [En línea]. Available: https://www.elecsaler.com/blog/industry/xt60-plugs-a-comprehensive-analysis-of-high-performance-connectivity-solutions-55010-1. [Último acceso: 14 Mayo 2024].

[48] «Liontec,» [En línea]. Available: https://www.liontec.es/producto/conector-xt60/.

[49] «Golden Motor,» [En línea]. Available: https://goldenmotor.bike/product/xt60-connector-set-with-pigtails/.

[50] «Melopero,» [En línea]. Available: https://www.melopero.com/es/tienda/componentes/circuitos-integrados/reguladores-de-voltaje/pololuajustablestepupregulador-de-voltajeu1v11a/. [Último acceso: 26 Junio 2024].

[51] A. Barrientos, Fundamentos de robótica, Mc Graw Hill.

[52] I. S. M. Khalil, «MNRLab,» - mayo 2016. [En línea]. Available: https://www.mnrlab.com/uploads/7/3/8/3/73833313/trajectory.pdf. [Último acceso: 5 Julio 2024].

[53] J. M. Fiore, «LibreTexts Engineering,» [En línea]. Available: https://eng.libretexts.org/Bookshelves/Electrical_Engineering/Electronics/Semiconductor_Devices_-_Theory_and_Application_(Fiore)/14%3A_Class_D_Power_Amplifiers/14.3%3A_Pulse_Width_Modulation. [Último acceso: 25 Junio 2024].

[54] «byjus,» [En línea]. Available: https://byjus.com/physics/pulse-width-modulation/. [Último acceso: 25 Junio 2024].

[55] «solectro,» [En línea]. Available: https://solectroshop.com/es/blog/que-es-pwm-y-como-usarlo--n38. [Último acceso: 25 Junio 2024].

[56] L. Harvie, «RunTime,» 19 Octubre 2023. [En línea]. Available: https://runtimerec.com/pulse-width-modulation/. [Último acceso: 25 Junio 2024].

[57] S. Dietrich, «Control Automation,» 23 marzo 2022. [En línea]. Available: https://control.com/technical-articles/understanding-the-basicsof-pulse-width-modulation-pwm/. [Último acceso: 25 Junio 2024].

[58] Yousef, «The Engineers Post,» 7 Abril 2022. [En línea]. Available: https://www.theengineerspost.com/servo-motor/. [Último acceso: 25 Junio 2024].

[59] P. Evans, «The Engineering Mindset,» 23 Enero 2022. [En línea]. Available: https://theengineeringmindset.com/servo-motor-xplained/. [Último acceso: 25 Junio 2024].

[60] W. Gastreich, «Realpars,» 27 Agosto 2024. [En línea]. Available: https://www.realpars.com/blog/servo-motor. [Último acceso: 25 Junio 2024].

[61] «NaylamP Mechatronics,» [En línea]. Available: https://naylampmechatronics.com/blog/33_tutorial-uso-de-servomotores-con-arduino.html. [Último acceso: 25 Junio 2024].

[62] S. Campbell, «Circuit Basics,» [En línea]. Available: https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/. [Último acceso: 25 Junio 2024].

[63] R. Keim, «All About Circuits,» 2015 Diciembre 2015. [En línea]. Available: https://www.allaboutcircuits.com/technical-articles/introduction-to-the-i2c-bus/. [Último acceso: 26 Junio 2024].

[64] «Prusa3D,» [En línea]. Available: https://www.prusa3d.com/es/. [Último acceso: 27 Junio 2024].

[65] «REE,» [En línea]. Available: https://www.ree.es/es. [Último acceso: 27 Junio 2024].

[66] S. Afzal, «Analog Devices,» 2 Septiembre 2016. [En línea]. Available: https://www.analog.com/en/resources/technical-articles/i2c-primer-what-is-i2c-part-1.html. [Último acceso: 26 Junio 2024].

[67] «Rip Max,» [En línea]. Available: http://www.ripmax.com/Item.aspx?ItemID=O-IP2850&Category=O0150. [Último acceso: 26 Junio 2024].

[68] L. A. Sánchez, «Apuntes de Robótica,» San Cristóbal de La Laguna, 2022.

10 Anexos

10.1 Señales PWM

10.1.1 Introducción a las señales PWM

El Pulse Width Modulation (PWM), o Modulación por Ancho de Pulso, es una técnica utilizada para controlar dispositivos electrónicos mediante señales digitales. Esta técnica permite ajustar la potencia media entregada a una carga variando la proporción de tiempo durante el cual la señal está en estado alto (encendida) en un ciclo completo de la señal.

La PWM es especialmente útil en aplicaciones que requieren un control preciso de potencia, velocidad y posición, como en motores, luces y sistemas de audio [53] [54].

10.1.2 Funcionamiento de las Señales PWM

Las señales PWM funcionan modulando el ancho de los pulsos de una señal digital para representar diferentes niveles de potencia. Una señal PWM se caracteriza por su ciclo de trabajo (duty cycle), que es la proporción del tiempo durante el cual la señal está en estado alto en un ciclo completo de la señal. El ciclo de trabajo se expresa en porcentaje y varía entre 0% (señal siempre apagada) y 100% (señal siempre encendida).

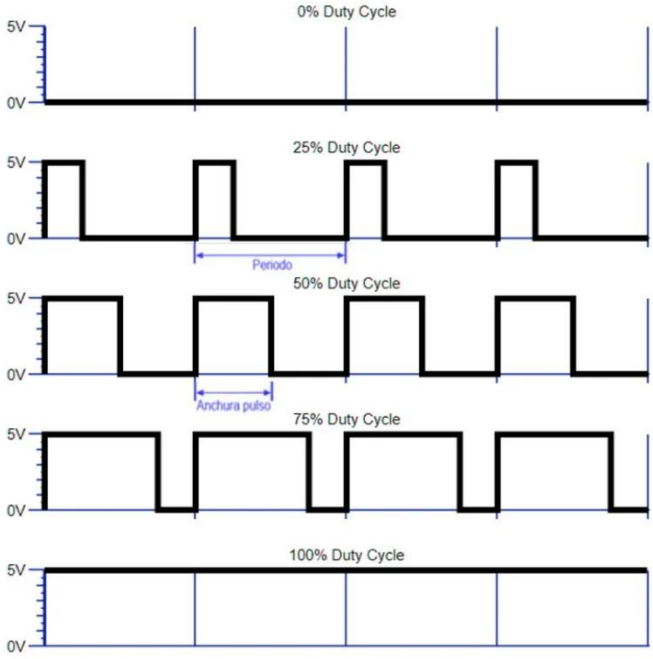


Figura 117. Diferentes señas PWM con distintos ciclos de trabajo. Fuente: [55]

Por ejemplo, un ciclo de trabajo del 50% significa que la señal está encendida la mitad del tiempo y apagada la otra mitad, proporcionando una potencia media. Al variar el ciclo de trabajo, se puede controlar de manera precisa la cantidad de potencia suministrada a una carga [53].

10.1.3 Generación de una Señal PWM

La señal PWM se genera utilizando un comparador. El modulating signal (señal de modulación) forma una parte de la entrada al comparador, mientras que la otra parte es una señal de onda no sinusoidal, como una onda de diente de sierra.

El comparador compara estas dos señales y genera una señal PWM como su salida. Si la señal de diente de sierra es mayor que la señal de modulación, la salida estará en estado alto; de lo contrario, estará en estado bajo. La anchura del pulso generado depende de la magnitud de la señal de modulación [54].

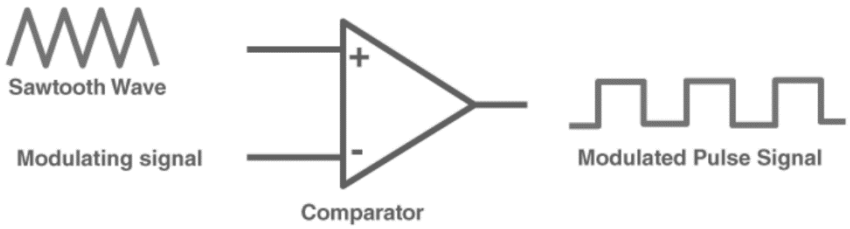


Figura 118. Generación de una señal PWM. Fuente: [54]

10.1.4 Parámetros Importantes de la Señal PWM

- Ciclo de Trabajo (Duty Cycle): Es el porcentaje de tiempo durante el cual la señal PWM está en estado alto. Se calcula como:

$$Ciclo\ de\ trabajo = \frac{Tiempo\ encendido}{Tiempo\ encendido + Tiempo\ apagado}$$

- Frecuencia: La frecuencia de la PWM determina cuántos ciclos completos (encendido + apagado) ocurren por segundo. Se calcula como:

$$Frecuencia = \frac{1}{Periodo\ de\ tiempo}$$

- Voltaje de Salida: El voltaje de salida promedio de la señal PWM será un porcentaje del voltaje de operación basado en el ciclo de trabajo. Por ejemplo, con un ciclo de trabajo del 50% y un voltaje de operación de 5V, el voltaje de salida será 2.5V [54].

10.1.5 Aplicaciones de las Señales PWM

Las señales PWM tienen una amplia gama de aplicaciones:

El control de motores permite ajustar tanto la velocidad como la posición de motores de corriente continua y servomotores, logrando esto mediante la variación del ciclo de trabajo [56].

En cuanto a la regulación del brillo en LEDs, se controla ajustando el ciclo de trabajo, lo cual es una técnica comúnmente utilizada en pantallas, retroiluminación e indicadores luminosos [56].

Para la generación de audio, estas técnicas son empleadas para producir señales de audio, resultandos útiles en la creación de tonos y la reproducción de audio [56].

En el ámbito de la carga de baterías, se controla la cantidad de potencia suministrada a las mismas durante su carga, regulando el voltaje de manera eficiente [57].

Finalmente, el control de ventiladores en computadoras se emplea para disipar el calor, ajustando la velocidad del ventilador mediante la modulación del ancho de pulso [57].

10.1.6 Ventajas del PWM

Las señales PWM ofrecen varias ventajas sobre otros métodos de control de potencia. En primer lugar, en términos de eficiencia energética, generan menos calor que los sistemas de control analógico, ya que los dispositivos de conmutación están mayormente en estados de saturación o corte, minimizando así las pérdidas de energía [54].

Además, permiten un control fino y preciso de la potencia entregada a la carga, siendo especialmente útiles en aplicaciones que requieren alta precisión.

Por último, su simplicidad de implementación es notable, ya que son fáciles de generar utilizando microcontroladores y otros dispositivos digitales, lo que simplifica el diseño de circuitos de control complejos [56] [54].

10.2 Servomotores

10.2.1 Introducción a los Servomotores

Los servomotores son motores eléctricos que permiten un control preciso de la posición angular, la velocidad y el par.

Estos motores son esenciales en una variedad de aplicaciones, desde la robótica y la automatización industrial hasta la aeronáutica y los sistemas de control de movimiento en tiempo real [58] [59].

10.2.2 Partes de un Servomotor

Un servomotor típico consta de varias partes clave:

- Motor: Generalmente un motor de corriente continua (DC) o alterna (AC) que proporciona el movimiento básico.
- Caja de Engranajes (Gearbox): Reduce la velocidad del motor y aumenta el par.
- Sensor de Posición (Potenciómetro): Proporciona retroalimentación sobre la posición actual del eje del motor.
- Controlador Electrónico: Procesa la señal de control y ajusta la posición del motor en consecuencia [60].
- Cojinetes y Ejes: Facilitan el movimiento suave y preciso del motor.
- Anillo de Retención (Snap Ring): Minimiza el daño al motor al mantener la posición del eje después de un golpe directo [58].
- Encoders: Dispositivos electromecánicos utilizados para transmitir la velocidad y la dirección del motor de vuelta al controlador [58].

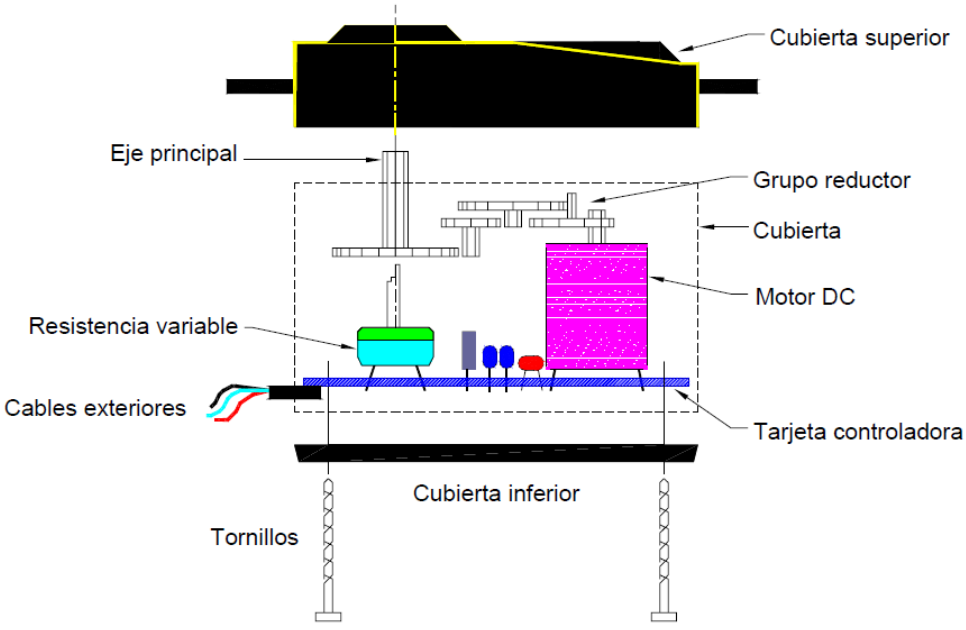


Figura 119. Partes de un servomotor. Fuente: [61]

10.2.3 Tipos de Servomotores

Los servomotores se pueden clasificar en varios tipos según su construcción y aplicación:

- DC Servo Motor: Utiliza fuentes de corriente continua y proporciona una respuesta rápida y precisa a las señales de control. Son comunes en máquinas de control numérico computarizado (CNC) y robótica [60].
- AC Servo Motor: Incluyen encoders y se utilizan en aplicaciones que requieren alta precisión y repetitividad, como la automatización industrial y maquinaria CNC [58] [60].
- Servo de Rotación Continua: Modificados para proporcionar control de velocidad y dirección en lugar de control de posición. Se usan comúnmente en robots móviles [58].
- Servo Lineal: Convierte el movimiento rotacional en movimiento lineal, utilizado en actuadores de precisión [58].

10.2.4 Servos Analógicos y Digitales

Los servos analógicos utilizan una señal de modulación por ancho de pulso (PWM) para determinar la posición del eje. Son más simples y económicos, pero menos precisos y más lentos que los digitales [60].

Por otro lado, los servos digitales incorporan un microcontrolador que procesa la señal de entrada y ajusta la posición del motor con mayor precisión y rapidez. Y ofrecen un mejor rendimiento bajo cargas variables [60]

10.2.5 Control de Servomotor

El control de un servomotor se realiza mediante señales PWM. La duración del pulso determina la posición angular del eje del motor. Por ejemplo, una señal PWM con un ciclo de trabajo del 50% podría mover el motor a una posición intermedia, mientras que un ciclo de trabajo del 100% lo movería a su posición máxima.

En un sistema típico, un controlador compara la posición deseada con la posición actual del motor (retroalimentada por el potenciómetro) y ajusta la señal PWM en consecuencia para minimizar el error. Este proceso se repite continuamente para mantener la precisión del control [60].

10.3 Comunicación I2C

10.3.1 Introducción a la Comunicación I2C

La comunicación I2C (Inter-Integrated Circuit) es un protocolo de comunicación serie sincrónico desarrollado por Philips Semiconductor (ahora NXP Semiconductors) en 1982.

Este protocolo está diseñado para permitir la comunicación entre múltiples dispositivos integrados en un mismo sistema, utilizando solo dos líneas de comunicación: una para la señal de datos (SDA) y otra para la señal de reloj (SCL) [62].

10.3.2 Funcionamiento del Protocolo I2C

El protocolo I2C es un sistema maestro-esclavo donde uno o más dispositivos maestros pueden controlar múltiples dispositivos esclavos a través de la misma línea de comunicación. Cada dispositivo esclavo tiene una dirección única de 7 bits (aunque también hay versiones de 10 bits) que permite al maestro identificar y comunicarse con ellos específicamente.

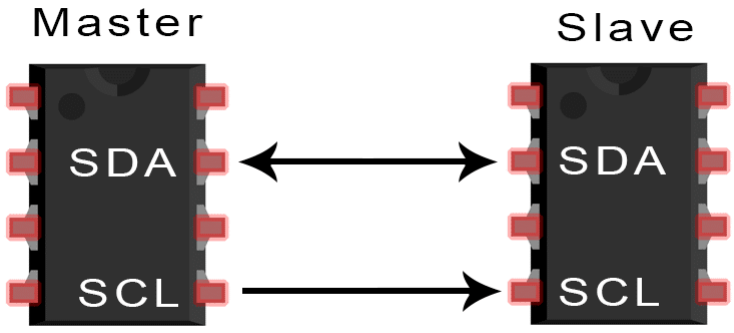


Figura 120. Dispositivo maestro controlando a un dispositivo esclavo usando el protocolo de comunicación I2C. Fuente: [62]

10.3.3 Operaciones Básicas del I2C:

- Condición de Inicio y Parada: La comunicación comienza con una condición de inicio (START) donde el maestro baja la línea SDA mientras SCL está alta. La comunicación se termina con una condición de parada (STOP) donde el maestro sube la línea SDA mientras SCL está alta.
- Dirección del Esclavo: Después de la condición de inicio, el maestro envía una dirección de 7 bits seguida por un bit de lectura/escritura. Los dispositivos esclavos monitorean el bus y responden solo si reconocen su dirección.
- Lectura y Escritura de Datos: Una vez que se establece la dirección, el maestro puede enviar o solicitar datos. Durante la transmisión de datos, el esclavo responde con un bit de reconocimiento (ACK) después de cada byte recibido para

confirmar la recepción correcta. Si el esclavo no reconoce los datos, envía un bit de no reconocimiento (NACK).

- Transmisión de Datos: Los datos se transmiten bit a bit, sincronizados con la señal de reloj. La línea SDA solo cambia de estado cuando SCL está baja, asegurando que los datos sean estables y se lean correctamente durante los flancos de subida del reloj.

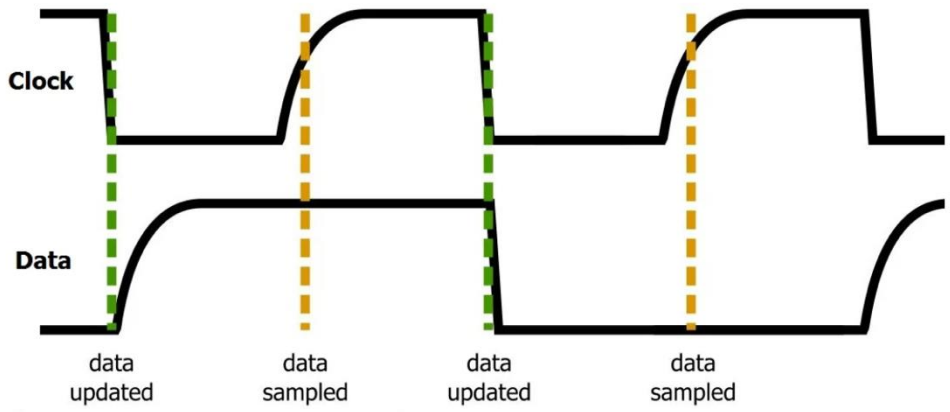


Figura 121. Diagrama de Temporización del Protocolo I2C: Actualización y Muestreo de Datos. Fuente: [63]

10.3.4 Ventajas y Desventajas del I2C

El protocolo ofrece una serie de ventajas y desventajas que lo hacen adecuado para ciertas aplicaciones, aunque con algunas limitaciones para tener en cuenta.

Entre sus ventajas, destaca el bajo número de pines necesarios. Al utilizar solo dos líneas de comunicación, simplifica notablemente el diseño de hardware y reduce el número de pines necesarios en los dispositivos, lo que puede ser decisivo en sistemas con espacio limitado o donde se desea minimizar la complejidad de las conexiones.

Otra ventaja importante es su flexibilidad. I2C soporta configuraciones con múltiples maestros y esclavos, lo que permite crear redes de comunicación flexibles y escalables. Esta capacidad es particularmente útil en sistemas donde varios dispositivos necesitan interactuar entre sí sin una estructura jerárquica rígida.

Además, el protocolo I2C incorpora mecanismos de control de errores, como los bits de reconocimiento (ACK/NACK). Estos bits aseguran que los datos se transmitan correctamente, proporcionando una capa adicional de fiabilidad en las comunicaciones entre dispositivos.

Sin embargo, el protocolo I2C también presenta algunas desventajas. Una de las más significativas es su velocidad limitada. Aunque puede alcanzar velocidades de hasta 3.4 Mbps en modos rápidos, sigue siendo más lento en comparación con otros protocolos como SPI (Serial Peripheral Interface), lo que puede ser una limitación en aplicaciones que requieren transferencias de datos a alta velocidad.

La complejidad también es una desventaja del I2C. La gestión de múltiples dispositivos y la resolución de condiciones de fallo requieren un manejo más complejo tanto en el firmware como en el hardware, lo que puede incrementar el esfuerzo de desarrollo y la probabilidad de errores.

Finalmente, el uso de resistencias pull-up es necesario para mantener las líneas SDA (data) y SCL (clock) en estado alto. Estas resistencias, aunque esenciales para el funcionamiento del protocolo, pueden aumentar el consumo de energía y el espacio necesario en la PCB, lo que puede ser una consideración importante en diseños donde se busca optimizar ambos factores [63].

10.3.5 Aplicaciones del I2C

El protocolo I2C se utiliza en una amplia variedad de aplicaciones, incluyendo:

1. Sensores: Comunicación con sensores de temperatura, presión, acelerómetros, etc.
2. Memorias: Interfaz con EEPROMs y otras memorias no volátiles.
3. Pantallas: Control de pantallas OLED, LCD y otros módulos de visualización.
4. Convertidores de Datos: Comunicación con ADCs (convertidores analógico-digitales) y DACs (convertidores digital-analógico).

10.3.6 Implementación y Configuración

Para implementar I2C en un proyecto, se deben seguir estos pasos:

1. Configuración del Bus: Establecer las líneas SDA y SCL con resistencias pull-up adecuadas.
2. Dirección de los Dispositivos: Asegurarse de que cada dispositivo esclavo tenga una dirección única.
3. Protocolo de Comunicación: Escribir el firmware para manejar las condiciones de inicio/parada, la dirección de los esclavos y la transmisión/recepción de datos.
4. Manejo de Errores: Implementar mecanismos para manejar errores de transmisión, como la falta de reconocimiento y la recuperación de fallos en el bus.

10.4 Cinemática en robots

La cinemática de robots se enfoca en el estudio del movimiento del robot respecto a un sistema de referencia sin considerar las fuerzas implicadas. En este campo, se busca describir analíticamente el movimiento espacial del robot a lo largo del tiempo, especialmente con relación a la posición y orientación del extremo final del robot según sus coordenadas articulares.

En la cinemática de robots se deben resolver dos problemas principales:

1. El problema cinemático directo, que consiste en determinar la posición y orientación del extremo final del robot a partir de sus coordenadas articulares y parámetros geométricos conocidos.
2. El problema cinemático inverso, que se encarga de hallar la configuración que debe adoptar el robot para alcanzar una posición y orientación específicas del extremo final.

Denavit y Hartenberg propusieron un método sistemático para describir y representar la geometría espacial de los eslabones de una cadena cinemática. Este método utiliza una matriz de transformación homogénea para describir la relación espacial entre elementos rígidos adyacentes, facilitando así la resolución del problema cinemático directo a través de una matriz de transformación homogénea 4×4 que relaciona la posición espacial del extremo del robot con el sistema de coordenadas base [51].

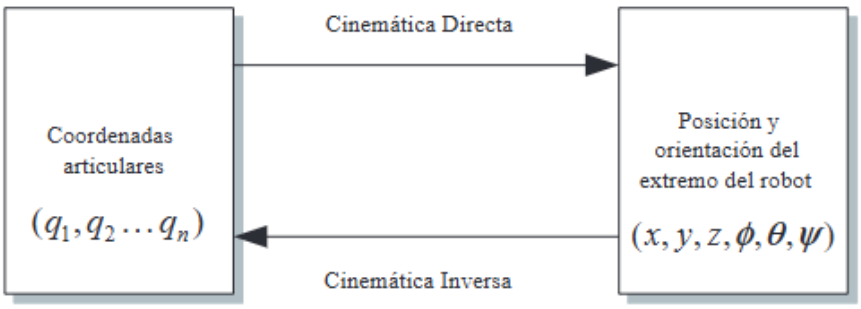


Figura 122. Diagrama de relación entre cinemática directa e inversa. Fuente: [51]

10.4.1 El problema cinemático directo

La resolución del problema cinemático directo permite determinar la posición y orientación del extremo del robot cuando los valores de sus variables articulares son conocidos. Las variables articulares, que pueden ser leídas por sensores y la unidad de control del robot, se utilizan para proporcionar información sobre la localización del extremo del robot al usuario.

La obtención del modelo cinemático directo puede abordarse mediante dos enfoques principales: métodos geométricos y métodos basados en cambios de sistemas de referencia.

Los métodos geométricos son adecuados para casos simples y robots con pocos grados de libertad, pero no son sistemáticos. En contraste, los métodos basados en cambios de sistemas de referencia permiten una aproximación sistemática para robots con cualquier número de grados de libertad. Entre estos, las matrices de transformación homogénea son frecuentemente utilizadas [51].

10.4.1.1 La resolución del problema cinemático directo mediante métodos geométricos

La resolución del problema cinemático directo consiste en encontrar las relaciones que permiten conocer la localización espacial del extremo del robot a partir de los valores de sus coordenadas articulares.

La obtención de estas relaciones puede ser, en ciertos casos (robots de pocos grados de libertad, GDL), ser fácil de encontrar mediante simples consideraciones geométricas. Por ejemplo, para los robots de 2GDL de la figura siguiente se puede obtener con facilidad:

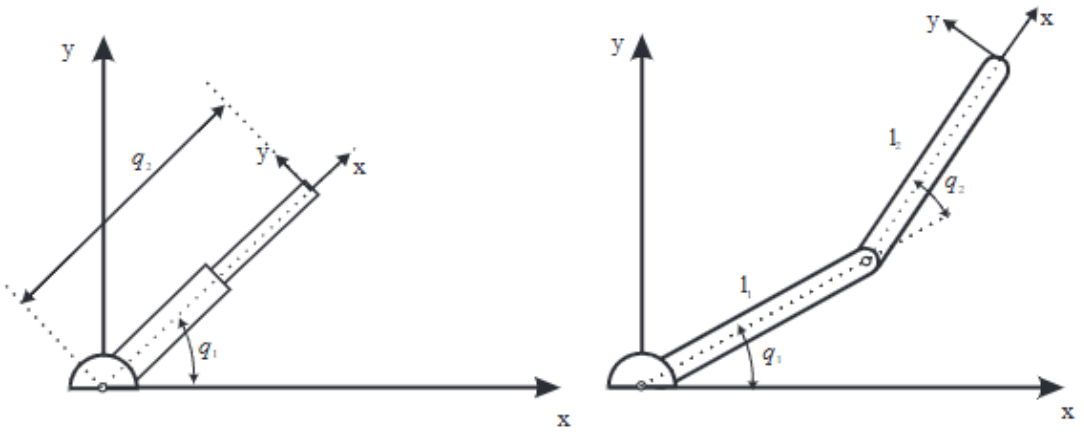


Figura 123. Robots planares de 2 grados de libertad. Fuente: [51]

Para el robot polar situado a la izquierda de la figura se tiene:

$$x = q_2 \cos q_1$$

$$y = q_2 \sin q_1$$

$$z = 0$$

Por otro lado para el robot articular a la derecha de la figura se tiene:

$$x = l_1 \cos q_1 + l_2 \cos(q_1 + q_2)$$

$$x = l_1 \sin q_1 + l_2 \mathrm{si\,n}(q_1 + q_2)$$

$$z = 0$$

Para un robot de 3 grados de libertad, como el de la siguiente figura en el que todos sus elementos quedan contenidos en un plano, puede trabajarse sobre éste, resultando un robot similar al de la figura anterior obteniéndose:

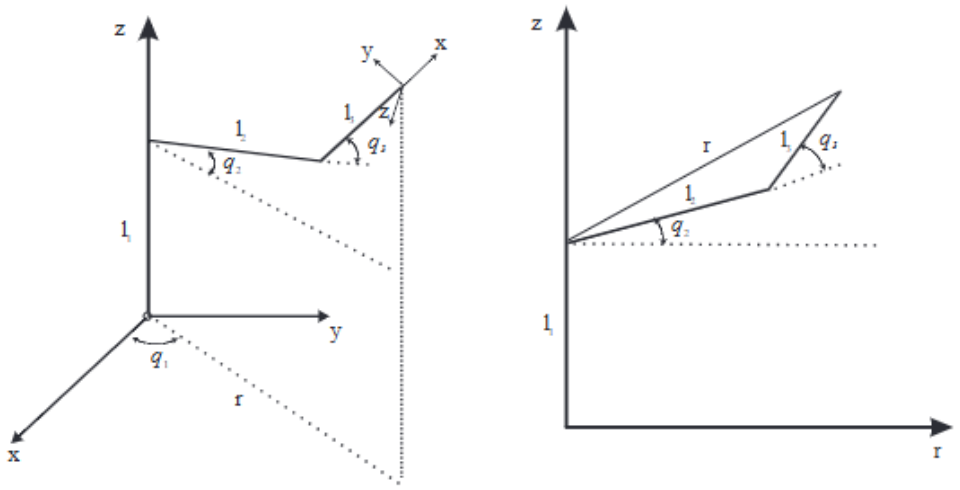


Figura 124. Robot de 3 grados libertad. Fuente: [51]

$$r = l_2 \cos q_2 + l_3 \cos(q_2 + q_3)$$

$$z = l_1 + l_2 \sin q_2 + l_3 \sin(q_2 + q_3)$$

$$x = r \cos q_1$$

$$y = r \sin q_1$$

No existe un procedimiento específico para resolver el problema cinemático directo mediante el método geométrico, resultando por ello, por lo general, inoperativo para robots de mayor número de grados de libertad. Para estos casos y de manera general, puede utilizarse el método basado en cambios de los sistemas de referencia, que se desarrollará en el apartado siguiente.

10.4.1.2 Resolución del problema cinemático directo mediante matrices de transformación homogénea

Como se ha visto en el apartado anterior, se utiliza fundamentalmente el álgebra vectorial y matricial para representar y describir la localización de un objeto en el espacio tridimensional con respecto a un sistema de referencia fijo. Dado que un robot se puede considerar como una cadena cinemática formada por objetos rígidos o eslabones unidos entre sí mediante articulaciones, se puede establecer un sistema de referencia fijo en la base del robot y describir la localización de cada uno de los eslabones con respecto a dicho sistema de referencia.

De esta forma, el problema cinemático directo se reduce a encontrar una matriz de transformación homogénea T que relacione la posición y orientación del extremo del robot respecto del sistema de referencia fijo situado en la base de este. Esta matriz T será función de las coordenadas articulares.

En general, un robot de n grados de libertad está formado por n eslabones unidos por n articulaciones, de forma que cada par articulación-eslabón constituye un grado de libertad. A cada eslabón se le puede asociar un sistema de referencia, utilizando las transformaciones homogéneas para representar las rotaciones y traslaciones relativas entre los distintos eslabones que componen el robot. Normalmente, la matriz de transformación homogénea que representa la posición y orientación relativa entre los sistemas asociados a dos eslabones consecutivos del robot se denomina matriz ${}^{i-1}A_i$.

Así pues, ${}^{0}A_1$ describe la posición y orientación del sistema de referencia solidario al primer eslabón con respecto al sistema de referencia solidario a la base, ${}^{1}A_2$ describe la posición y orientación del segundo eslabón respecto del primero, etc. Del mismo modo, denominando ${}^{0}A_i$ a las matrices resultantes del producto de las matrices ${}^{i-1}A_i$ con i desde 1 hasta k, se puede representar de forma total o parcial la cadena cinemática que forma el robot.

$$ {}^{0}A_2 = {}^{0}A_1 A_2 $$

De manera análoga, la matriz ${}^{0}A_3$representa la localización del sistema del tercer eslabón:

$$ {}^{0}A_3 = {}^{0}A_1 A_2 A_3 $$

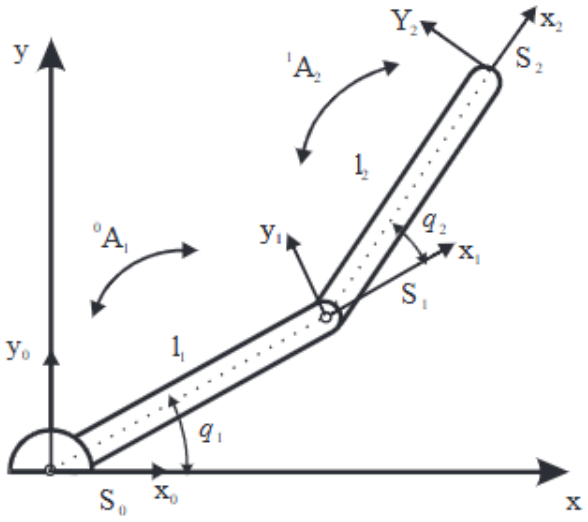


Figura 125. Asociación de sistemas de referencia a cada eslabón del robot. Fuente: [51]

Cuando se consideran todos los grados de libertad, a la matriz ${}^0A_n$ se le suele denominar *T*. Así, dado un robot de seis grados de libertad, se tiene que la posición y orientación del eslabón final vendrá dada por la matriz:

$$T = {}^0A_6 = {}^0A_1A_2A_3A_4A_5A_6$$

Cada una de las matrices ${}^{i-1}A_i$ representa el cambio de base que permite pasar del sistema asociado al eslabón $i-1$ al asociado al eslabón i. Esta matriz dependerá, además de constantes geométricas propias del eslabón, del grado de libertad q_i. Por lo tanto, la expresión anterior podrá escribirse como:

$$T(q_1, \dots, q_n) = {}^0A_1(q_1) \cdot {}^1A_2(q_2) \cdots {}^{n-1}A_n(q_n)$$

Resultando que la relación entre el sistema de coordenadas de la base y del extremo queda definida por una matriz de transformación homogénea *T* función de las coordenadas articulares, que debe hacerse coincidir con la matriz de transformación no homogénea correspondiente a la localización en la que se desea posicionar al robot. De esta igualdad se obtiene la solución al problema cinemático directo [51].

10.4.1.3 Algoritmo de Denavit-Hartenberg para la obtención del modelo cinemático directo

Para describir la relación entre elementos contiguos de un robot, se suele utilizar el método matricial de Denavit-Hartenberg (D-H). Este método, propuesto en 1955, establece la localización del sistema de coordenadas ligado a cada eslabón en una cadena articulada, permitiendo sistematizar la obtención de las ecuaciones cinemáticas.

Utilizando sistemas de coordenadas asociados a cada eslabón, D-H permite describir las relaciones espaciales mediante cuatro transformaciones básicas:

1. Rotación alrededor del eje z_{i-1} un ángulo θ_i.
2. Traslación a lo largo de z_{i-1} una distancia d_i.
3. Traslación a lo largo de x_i una distancia a_i.
4. Rotación alrededor del eje x_i un ángulo α_i.

Estas transformaciones permiten relacionar el sistema de referencia del eslabón $i-1$ con el sistema del eslabón i. Las matrices de transformación homogénea resultantes de obtienen mediante el producto de matrices en el orden adecuado:

$${}^{i-1}A_i = Rotz(\theta_i) \cdot T(0,0,d_i) \cdot T(a_i,0,0) \cdot Rotx(\alpha_i)$$

Realizando el producto de matrices se obtiene:

$$ {}^{i-1}A_i = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & S\alpha_i & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} = $$

$$ \begin{bmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

Donde $\theta_i, d_i, a_i, \alpha_i$ son los parámetros D-H del eslabón i. D este modo, basta con identificar los estos parámetros para obtener las matrices ${}^{i-1}A_i$, y así relacionar todos y cada uno de los eslabones del robot. En concreto estos representa:

θ_i Es el ángulo que forma los ejes x_{i-1} y x_i medido en un plano perpendicular al eje z_{i-1}, utilizando la regla de la mano derecha. Se trata de un parámetro variable en articulaciones giratorias.

d_i Es la distancia a lo largo del eje z_{i-1} desde el origen del sistema de coordenadas (i-1)-ésimo hasta la intersección de eje z_{i-1} con el eje x. Se tratade un parámetro variable en articulaciones prismáticas.

a_i Es la distancia a lo largo del eje x_i que va desde la intersección del eje z_{i-1} con el eje x_i hasta el origen del sistema i-ésimo, en el caso de articulaciones giratorias. En el cao de articulaciones prismáticas, se calcula como la distancia más corta entre los ejes z_{i-1} y z_i.

α_i Es el ángulo de separación del eje z_{i-1} y el eje z_i, medido en un plano perpendicular al eje x, utilizando la regla de la mano derecha.

Una vez obtenidos los parámetros D-H, el cálculo de las relaciones entre los eslabones consecutivos del robot es inmediato ya que vienen dadas por las matrices ${}^{i-1}A_i$. Las relaciones entre varios eslabones consecutivos dos a dos vienen dadas por la matrices T que, como ya se comentó anteriormente, se obtienen como producto de un conjunto de matrices A.

Obtenida la matriz T, esta expresará la orientación y posición del extremo del robot en función de sus coordenadas articulares, con lo que quedará resuelto el problema cinemático directo [51].

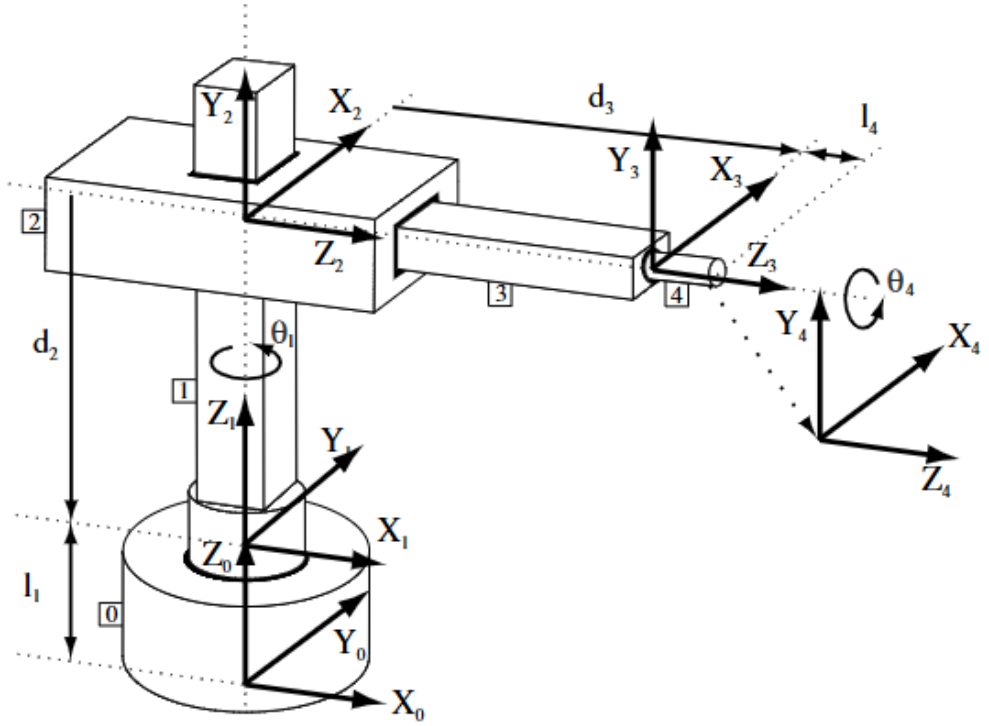


Figura 126. Robot cilíndrico. Fuente: [51]

10.4.2 Cinemática inversa

El objetivo del problema cinemático inverso consiste en encontrar los valores que deben adoptar las coordenadas articulares del robot $q = [q_1, q_2, \dots, q_n]^T$ para que su extremo se posicione y oriente según una determinada localización espacial $(p, [n, o, a])$.

Así cómo es posible abordar el problema cinemático directo de una manera sistemática a partir de la utilización de matrices de transformación homogéneas, e independientemente de la configuración del robot, no ocurre lo mismo con el problema cinemático inverso, siendo el procedimiento de obtención de las ecuaciones fuertemente dependiente de la configuración del robot.

Se han desarrollado algunos procedimientos genéricos susceptibles de ser programados de modo que un computador pueda, a partir del conocimiento de la cinemática del robot (con sus parámetros de Denavit-Hartenberg, por ejemplo) obtener la n-tupla de valores articulares que posicionan y orientan su extremo.

El inconveniente de estos procedimientos es que se trata de métodos numéricos iterativos, cuya velocidad de convergencia e incluso su convergencia en sí no está siempre garantizada.

A la hora de resolver el problema cinemático inverso es mucho más adecuado encontrar una solución cerrada. Esto es, encontrar una relación matemática explícita de la forma:

$$q_k = f_k(x, y, z, \phi, \theta, \psi)$$

$$k = 1, \dots, n(GDL)$$

Este tipo de solución presenta, entre otras, las siguientes ventajas:

1. En muchas aplicaciones, el problema cinemático inverso ha de resolverse en tiempo real (por ejemplo, en el seguimiento de una determinada trayectoria). Una solución de tipo iterativo no garantiza tener la solución en el momento adecuado.
2. Al contrario de lo que ocurriría en el problema cinemático directo, con cierta frecuencia la solución del problema cinemático inverso no es única; existiendo diferentes n-tuplas $[q_1, \dots, q_n]^T$ que posicionan y orientan el extremo del robot del mismo modo. En estos casos una solución cerrada permite incluir determinadas reglas o restricciones que aseguren que la solución obtenida sea la más adecuada de entre las posibles (por ejemplo, límites en los recorridos articulares).

No obstante, a pesar de las dificultades comentadas, la mayor parte de los robots poseen cinemáticas relativamente simples que facilitan en cierta medida la resolución de su problema cinemático inverso. Por ejemplo, si se consideran sólo los tres primeros grados de libertad de muchos robots, éstos tienen una estructura planar, es decir, los tres primeros elementos quedan contenidos en un plano. Esta circunstancia simplifica la resolución del problema. Así mismo, en muchos robots se da la circunstancia de que los ejes de los últimos grados de libertad, dedicados fundamentalmente a orientar el extremo del robot, corresponden a giros sobre ejes que se cortan en un punto. De nuevo esta situación facilita el cálculo de la n-tupla $[q_1, \dots, q_n]^T$ correspondiente a la posición y orientación deseadas. Por tanto, para los casos citados y otros, es posible establecer ciertas pautas generales que permitan plantear y resolver el problema cinemático inverso de una manera sistemática.

Los métodos geométricos permiten, normalmente, obtener los valores de las primeras variables articulares, que son las que consiguen posicionar el robot (prescindiendo de la orientación de su extremo). Para ello utilizan relaciones trigonométricas y geométricas sobre los elementos del robot. Se suele recurrir a la resolución de triángulos formados por los elementos y articulaciones del robot.

Como alternativa para resolver el mismo problema se puede recurrir a manipular directamente las ecuaciones correspondientes al problema cinemático directo. Es decir, puesto que éste establece la relación:

$$\begin{bmatrix} n & o & a & p \\ 0 & 0 & 0 & 0 \end{bmatrix} = [t_{ij}]$$

donde los elementos t_{ij} son función de las coordenadas articulares $[q_1, \dots, q_n]^T$, es posible pensar que mediante ciertas combinaciones de las 12 ecuaciones planteadas en anteriormente se puedan despejar las n variables articulares q_i en función de las componentes de los vectores n, o, a y p. Debe considerarse en este caso que en general las 12 ecuaciones responden a ecuaciones trigonométricas acopladas cuya resolución no es trivial.

Para facilitar esta solución se verá que se puede proceder de manera ordenada, despejando sucesivamente los grados de libertad.

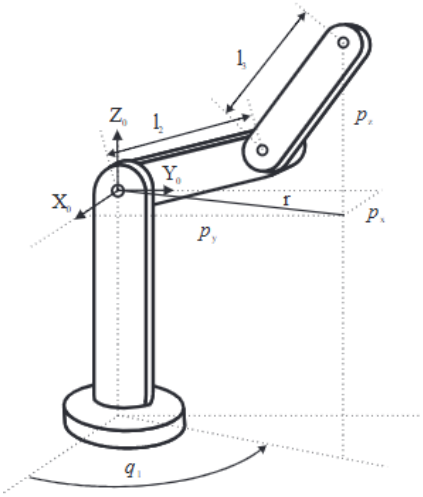


Figura 127. Robot articular. Fuente: [51]

Por último, si se consideran robots con capacidad de posicionar y orientar su extremo en el espacio, esto es, robots con 6 GDL, el método de desacoplamiento cinemático permite, para determinados tipos de robots, resolver los primeros grados de libertad, dedicados al posicionamiento, de manera independiente a la resolución de los últimos grados de libertad, dedicados a la orientación. Cada uno de estos dos problemas más simples podrá ser tratado y resuelto por cualquiera de los procedimientos anteriores [51].

10.4.2.1 Resolución del problema cinemático inverso por métodos geométricos

Como se ha indicado, este procedimiento es adecuado para robots de pocos grados de libertad o para el caso de que se consideren sólo los primeros grados de libertad, dedicados a posicionar el extremo.

El procedimiento en sí se basa en encontrar suficiente número de relaciones geométricas en las que intervendrán las coordenadas del extremo del robot, sus coordenadas articulares y las dimensiones físicas de sus elementos.

Para mostrar el procedimiento a seguir se va a aplicar el método a la resolución del problema cinemático inverso de un robot con 3 GDL de rotación (estructura típica articular). La figura siguiente muestra la configuración del robot. Los datos de partida son las coordenadas (p_x, p_y, p_z) referidas a $\{S_0\}$ en las que se quiere posicionar su extremo.

Como se ve, este robot posee una estructura planar, quedando este plano definido por el ángulo de la primera variable articular q_1.

El valor de q_1 se obtiene inmediatamente como:

$$q_1 = arctg\left(\frac{p_y}{p_x}\right)$$

Considerando ahora únicamente los elementos 2 y 3 que están situados en un plano, y utilizando el teorema del coseno, se tendrá:

$$\begin{cases} r^2 = {p_x}^2 + {p_y}^2 \\ r^2 + {p_z}^2 = {l_2}^2 + {l_3}^2 + 2l_2 l_3 \cos q_3 \end{cases}$$

$$cosq_3 = \frac{{p_x}^2 + {p_y}^2 + {p_z}^2 - {l_2}^2 - {l_3}^2}{2l_2l_3}$$

Esta expresión permite obtener q_3 en función del vector de posición del extremo. No obstante, y por motivos de ventajas computacionales, es más conveniente utilizar la expresión del arco tangente en lugar del arco coseno.

Puesto que:

$\sin q_3 = \pm\sqrt{1 - cos^2 q_3}$ tendrá que:

$$q_3 = arctg\left(\frac{\pm\sqrt{1-cos^2 q_3}}{cosq_3}\right)\text{n}$$

$$cosq_3 = \frac{{p_x}^2 + {p_y}^2 + {p_z}^2 - {l_2}^2 - {l_3}^2}{2l_2l_3}$$

Como se ve, existen dos posibles soluciones para q_3 según se tome el signo positivo o el signo negativo en la raíz. Estas corresponden a las configuraciones de codo arriba y codo abajo del robot.

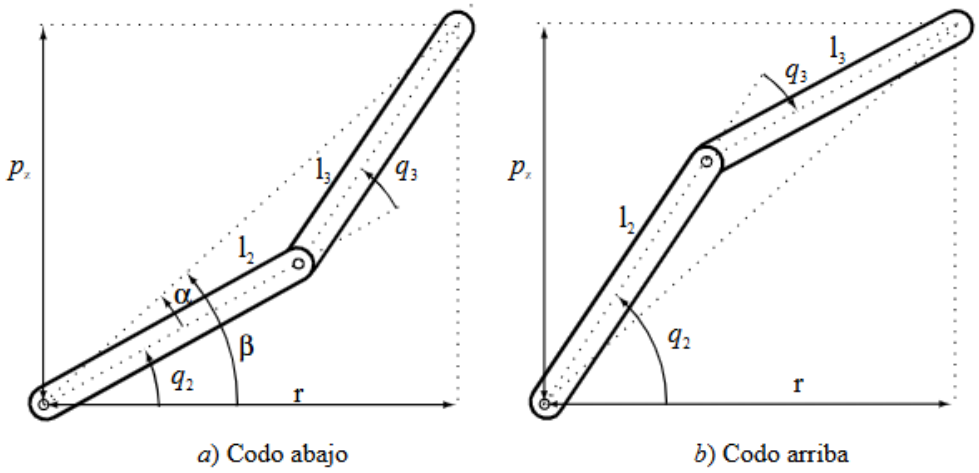


Figura 128. Elementos 2 y 3 del robot contenidos en un plano y en configuración codo abajo a la izquierda y configuración codo arriba a la derecha. Fuente: [51]

El cálculo de q_2 se hace a partir de la diferencia entre β y α:

Siendo:

$$\beta = arctg\left(\frac{p_z}{r}\right) = arctg\left(\frac{p_z}{\pm\sqrt{{p_x}^2 + {p_y}^2}}\right)$$

$$\alpha = arctg\left(\frac{l_3 \sin q_3}{l_2 + l_3 \cos q_3}\right)$$

Luego, finalmente:

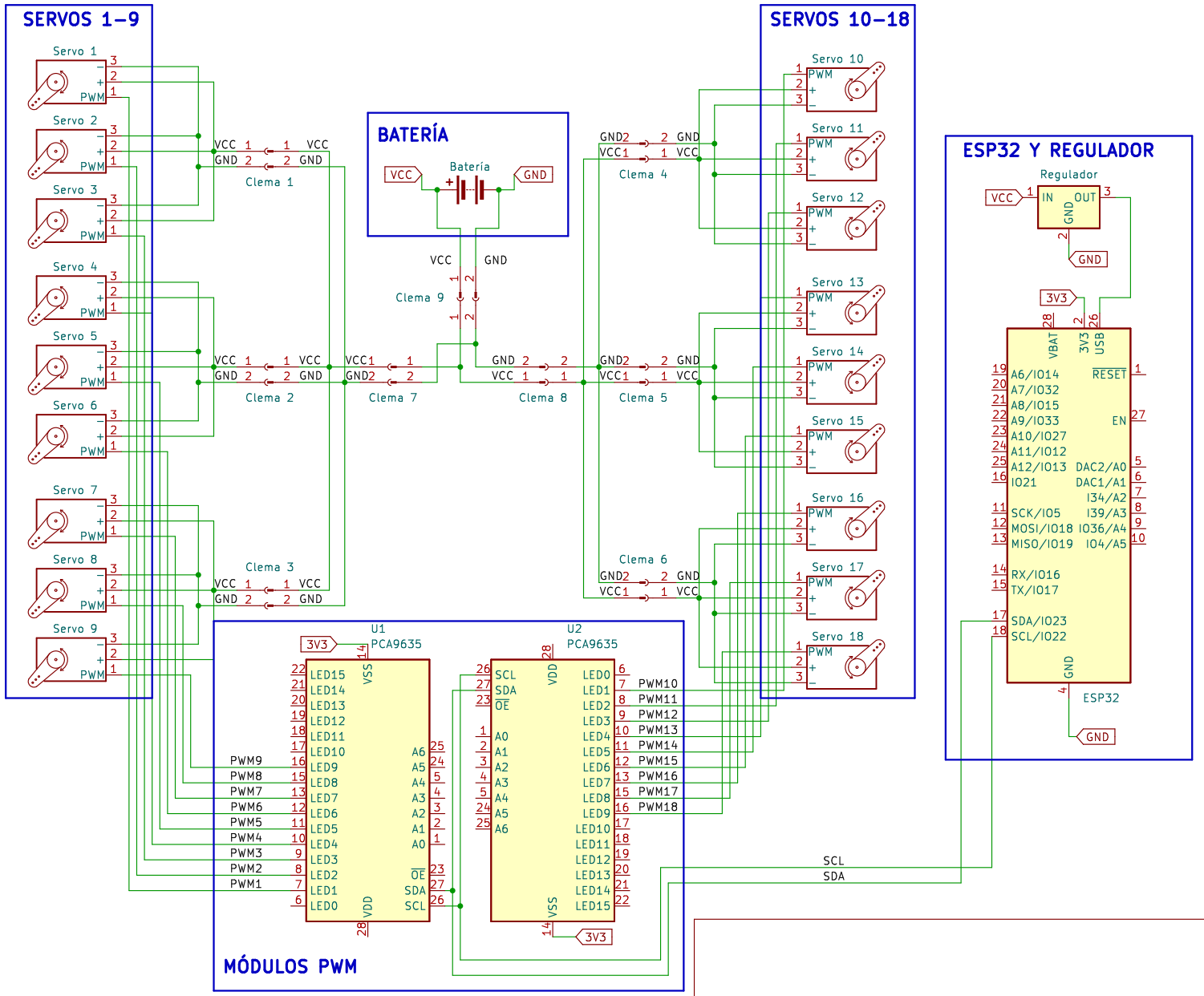
$$q_2 = arctg\left(\frac{p_z}{\pm\sqrt{{p_x}^2 + {p_y}^2}}\right) - arctg\left(\frac{l_3 \sin q_3}{l_2 + l_3 \cos q_3}\right)$$

De nuevo los dos posibles valores según la elección del signo dan lugar a dos valores diferentes de q_2 correspondientes a las configuraciones codo arriba y abajo.

Las Expresiones anteriores resuelven el problema cinemático inverso para el robot de 3 GDL considerado [51].

10.5 Esquema de Conexiones

A continuación, se muestra el esquema del cableado del robot hexápodo realizado en KiCad. En este diagrama se pueden observar las conexiones detalladas de los servomotores, módulos PWM, ESP32 y batería, así como los puntos de soldadura y la distribución de los cables.



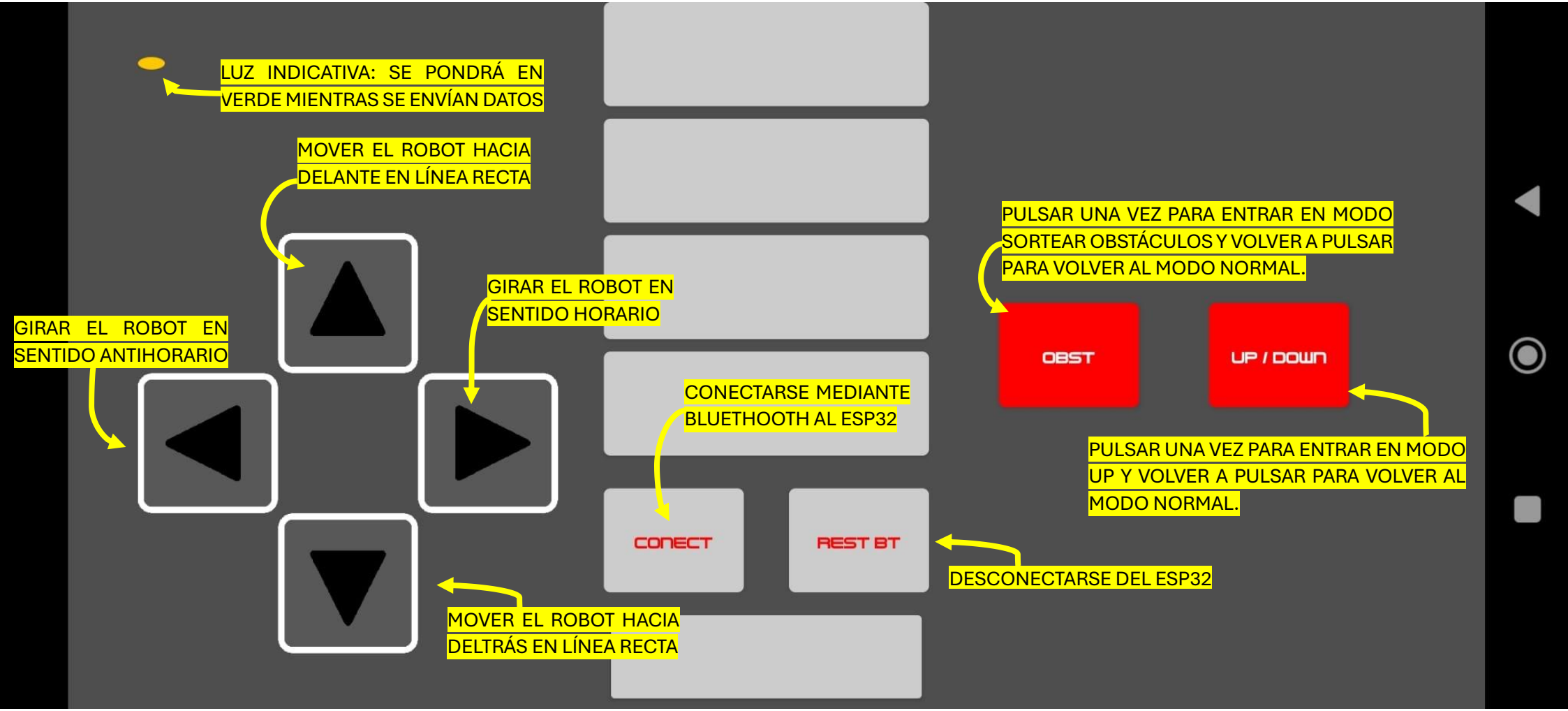
Sheet: /	**Autores:** Andres Jorges Palenzuela y Fabrizio Rodríguez Betancourt
File: HEXA.kicad_sch	
Title:	**Cableado y conexionado del hardaware electrónico del robot hexápodo**
Size: A4	Date: 04/07/2024
KiCad E.D.A. kicad 7.0.8	**Rev:** Id: 1/1

10.6 Manual de usuario para la aplicación móvil

Se muestra el manual se usuario con la interfaz de la aplicación y la función de cada uno de los botones.

Link para descargar la App.

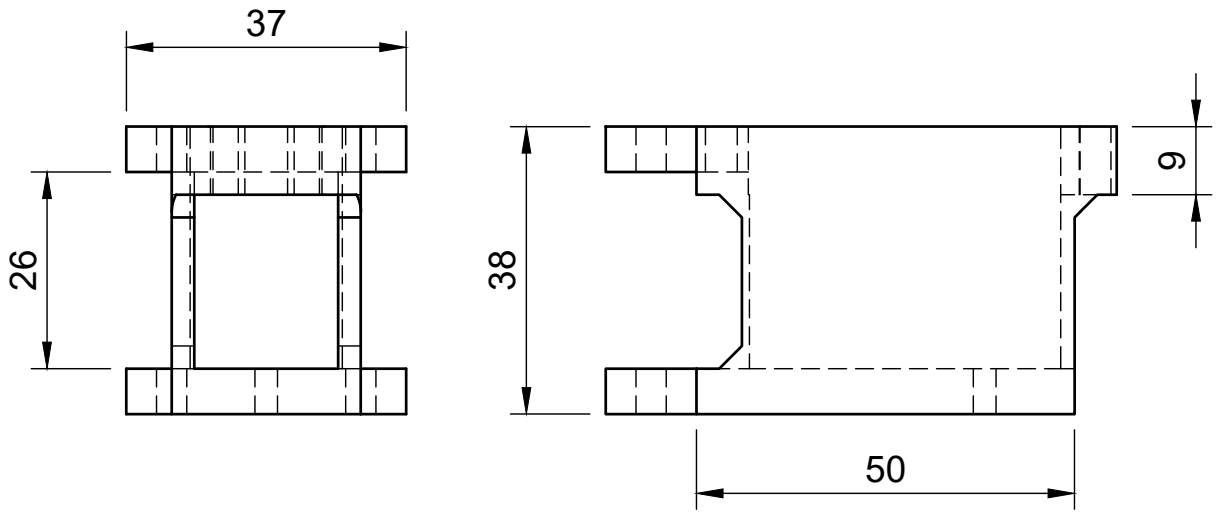
MANUAL DE USUARIO DE LA APLICACIÓN PARA MOVER EL ROBOT HEXÁPODO (Hexa_Remote_Control)

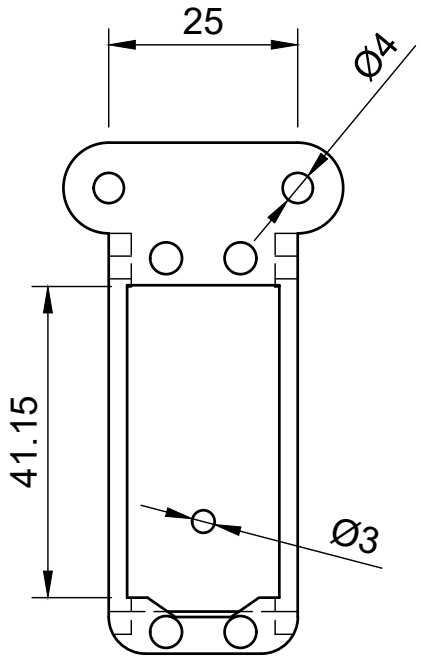


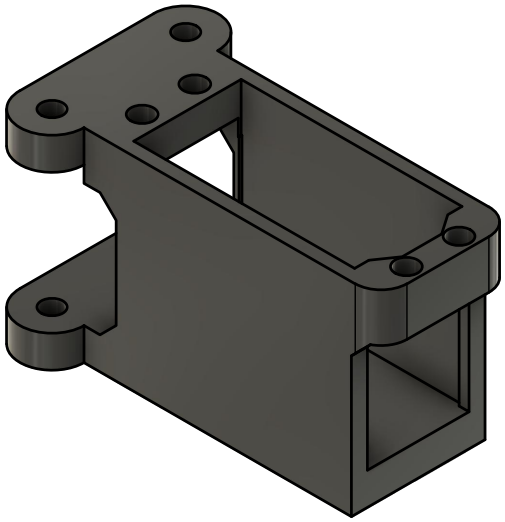
MANUAL DE USUARIO DE LA APLICACIÓN PARA MOVER EL ROBOT HEXÁPODO (Hexa_Remote_Control)

10.8 Diagramas y planos de diseño

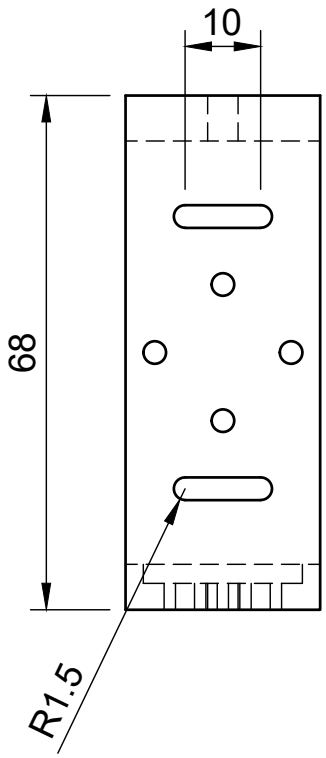
A continuación se muestran las vistas de cada una de las piezas y componentes usadas en CAD con sus respectivas cotas y dimensiones:

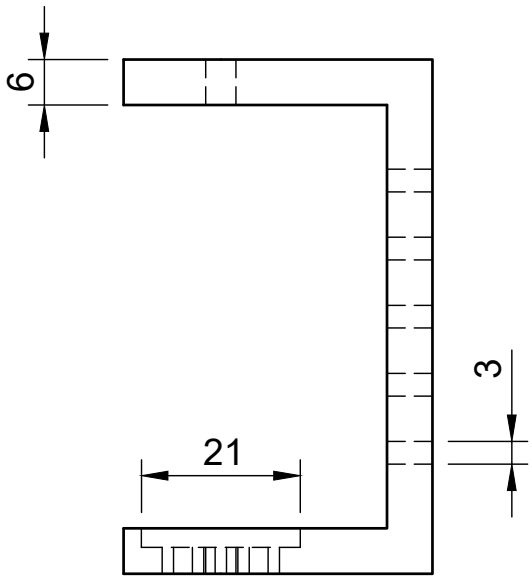


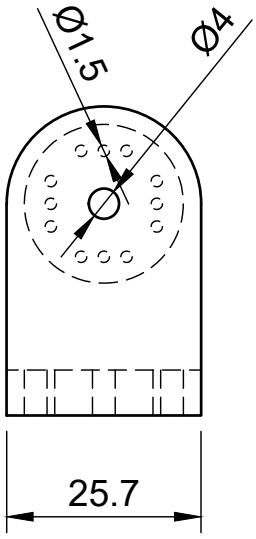


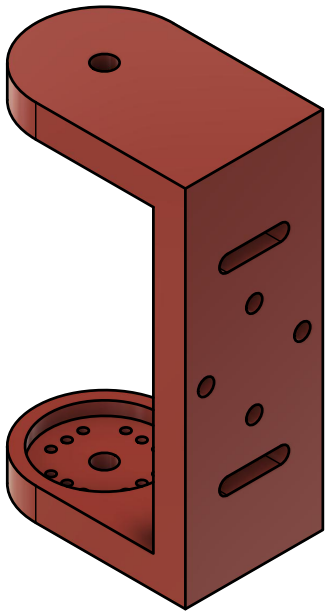


Dept.	Technical reference	Created by	Approved by
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024	
Escala 1:1		Document type	Document status
		Title: Caja Servo	DWG No.
			Rev. / Date of issue / Sheet: 1/1

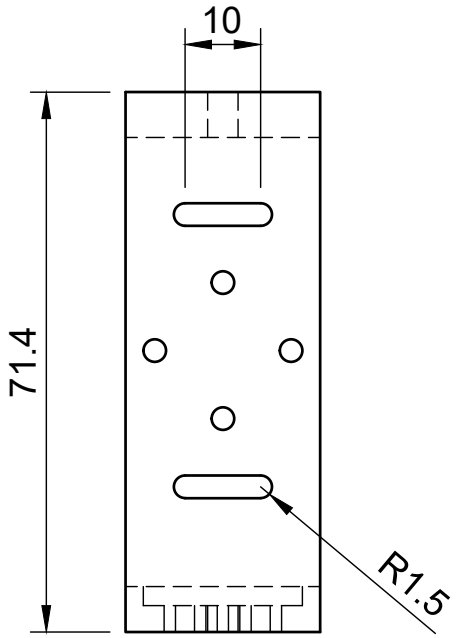


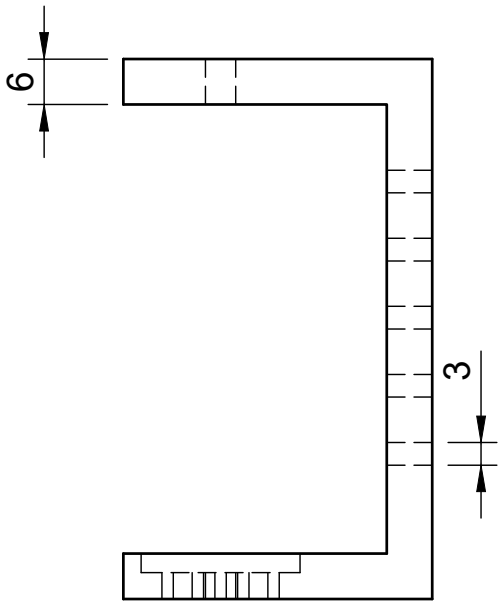


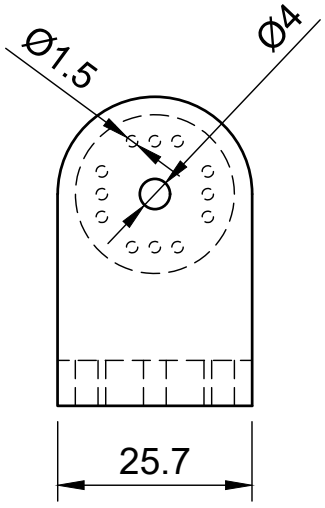


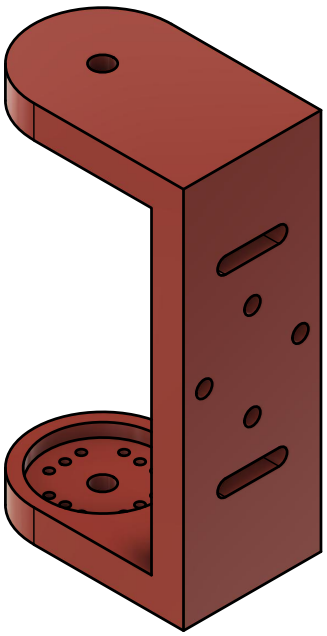


Dept.	Technical reference	Created by	Approved by	
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024		
Escala 1:1		Document type	Document status	
		Title U 68	DWG No.	
			Rev.	Date of issue / Sheet 1/1

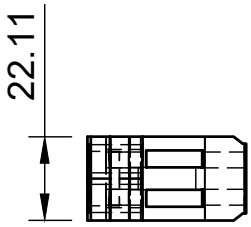


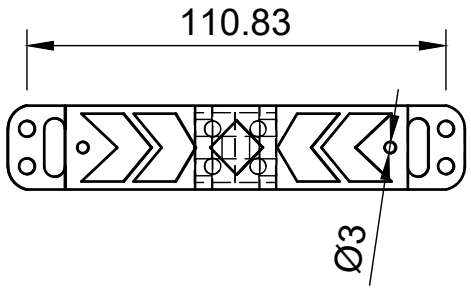


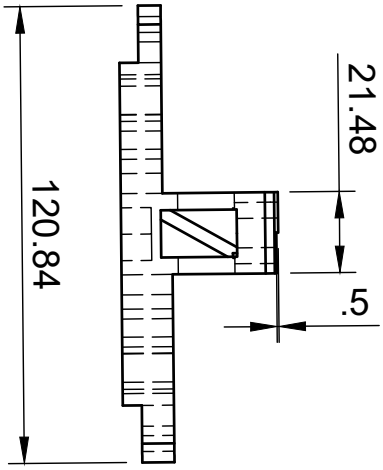


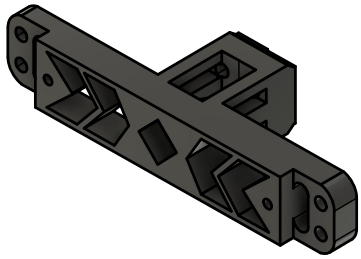


Dept.	Technical reference	Created by	Approved by
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024	
Escala 1:1		Document type	Document status
		Title U 714	DWG No.
			Rev. / Date of issue / Sheet 1/1

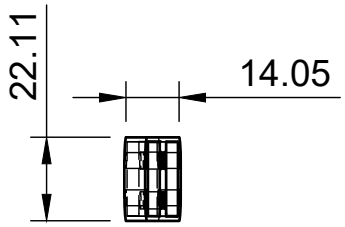


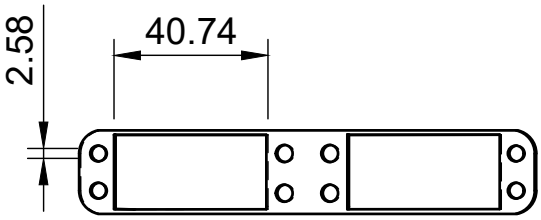


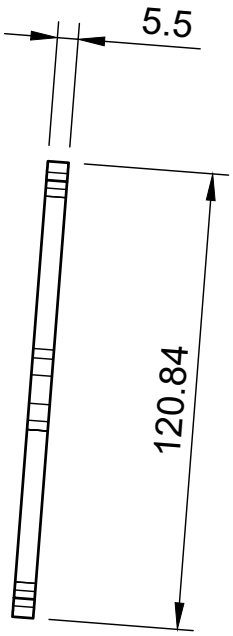


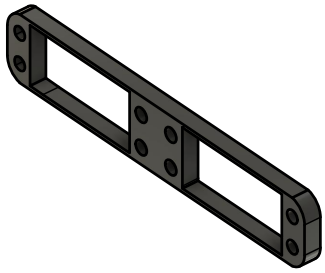


Dept.	Technical reference	Created by	Approved by	
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024		
Escala 1:2		Document type	Document status	
		Title Femur (parte inferior)	DWG No.	
			Rev. / Date of issue	Sheet 1/1

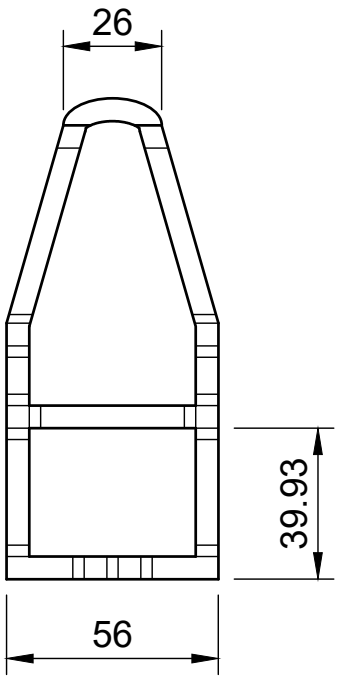


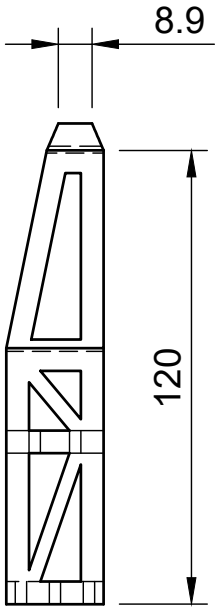


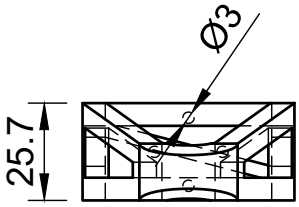


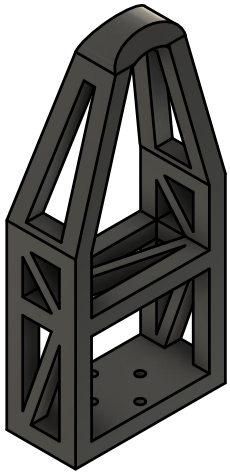


Dept.	Technical reference	Created by	Approved by
	Dimensiones mm	Fabrizio Rodríguez 23/06/2024	
Escala 1:2		Document type	Document status
		Title: Femur (parte superior)	DWG No.
			Rev. / Date of issue / Sheet: 1/1

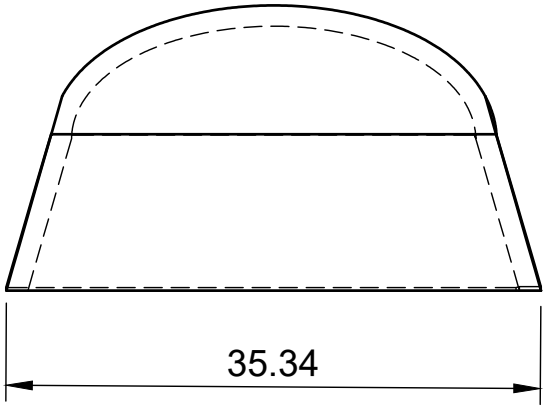


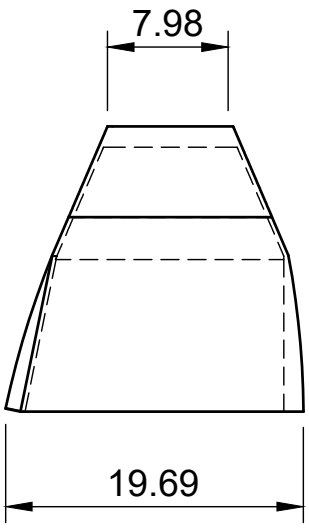


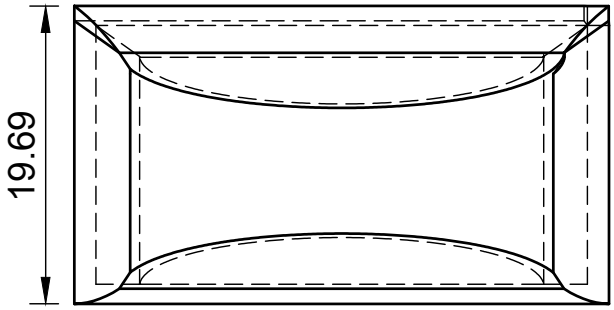


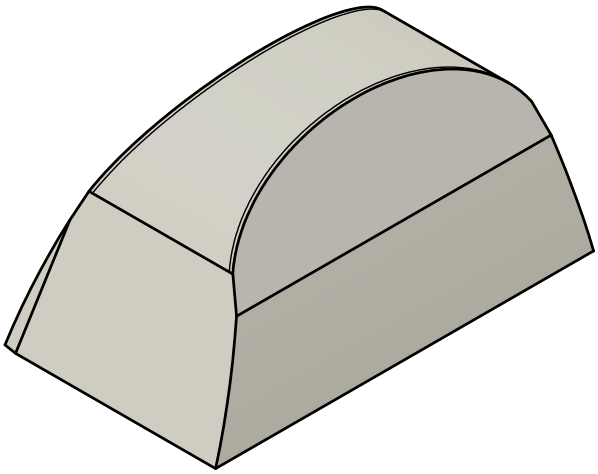


Dept.	Technical reference	Created by	Approved by
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024	
Escala 1:2		Document type	Document status
		Title: tibia ligera	DWG No.
			Rev. / Date of issue / Sheet: 1/1

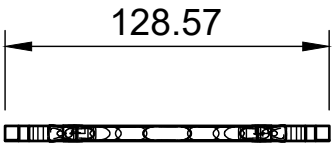


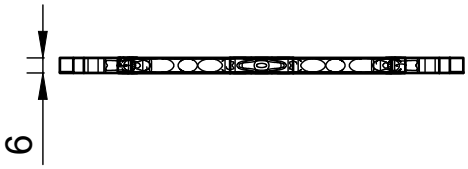


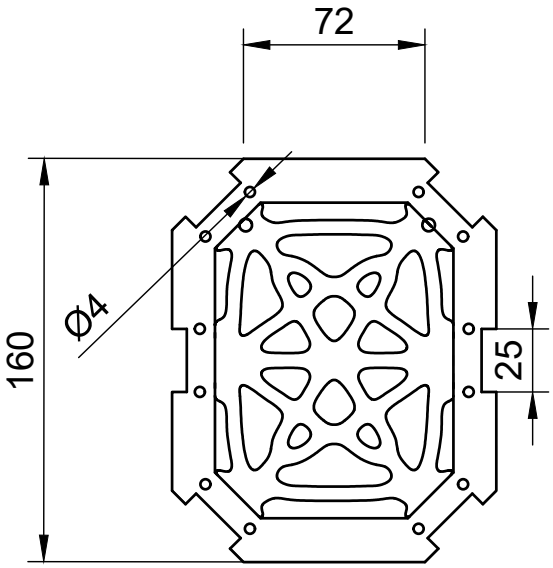


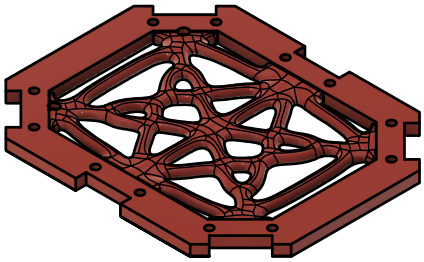


Dept.	Technical reference	Created by	Approved by	
	Dimensiones en mm	Fabrizio Rodríguez 27/06/2024		
Escala 2:1		Document type	Document status	
		Title tibia hembra	DWG No.	
			Rev.	Date of issue / Sheet 1/1

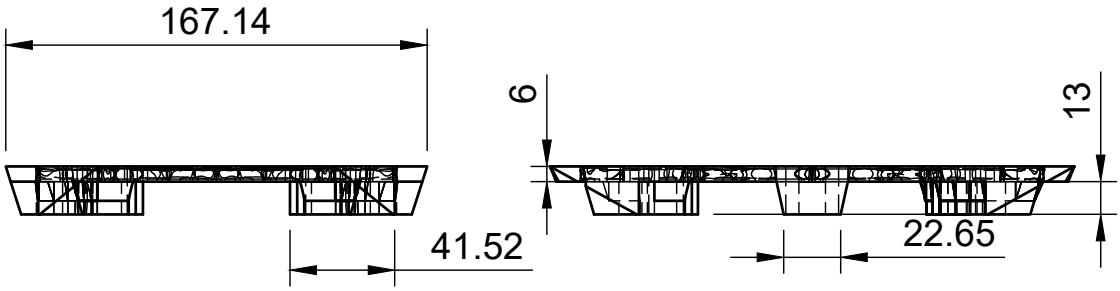


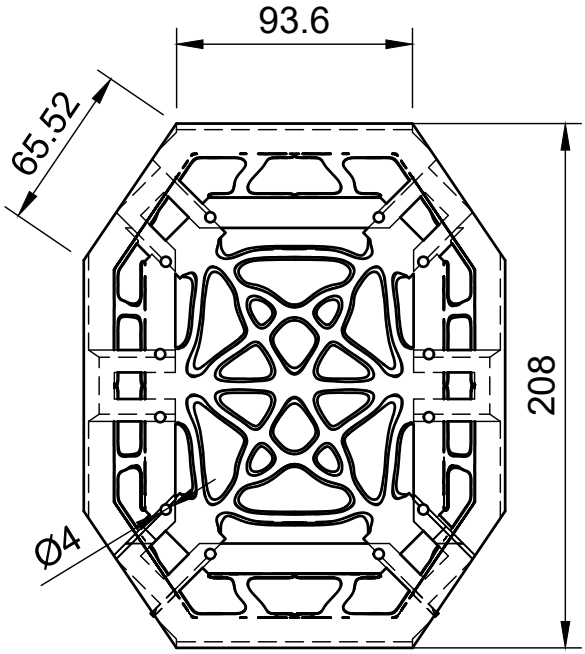


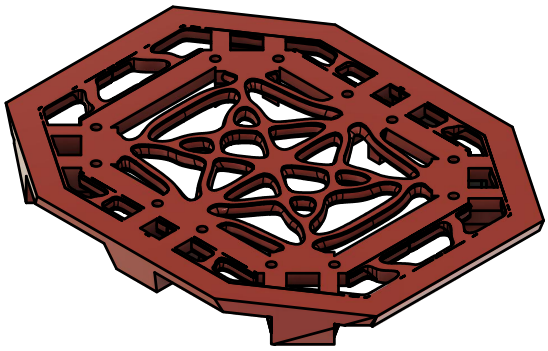




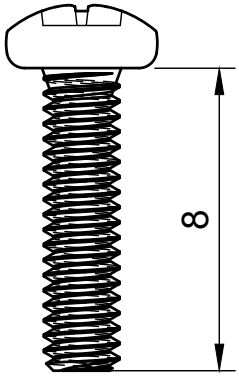
Dept.	Technical reference	Created by	Approved by
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024	
Escala 1:3		Document type	Document status
		Title: Cuerpo (Parte Abajo)	DWG No.
			Rev. / Date of issue / Sheet: 1/1

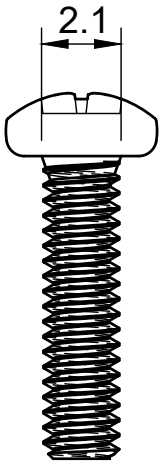


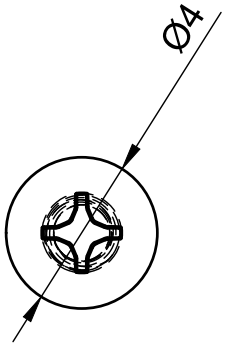


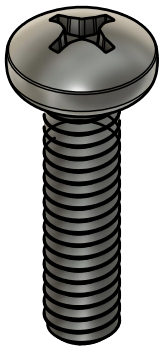


Dept.	Technical reference	Created by	Approved by
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024	
Escala 1:3		Document type	Document status
		Title: Cuerpo (Parte Arriba)	DWG No.
			Rev. / Date of issue / Sheet: 1/1

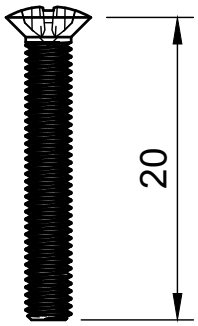


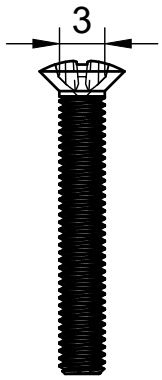


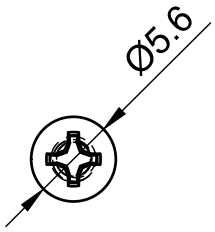


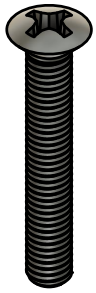


Dept.	Technical reference	Created by	Approved by	
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024		
Escala 5:1		Document type	Document status	
		Title M2 8mm	DWG No.	
			Rev.	Date of issue / Sheet 1/1

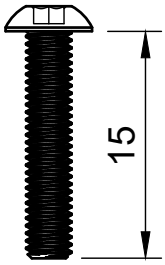


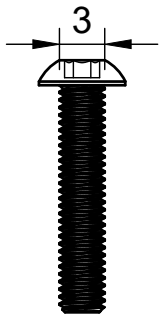


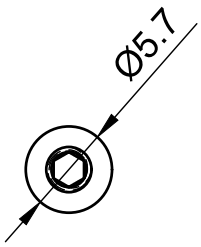


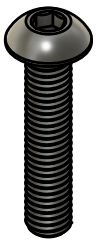


Dept.	Technical reference	Created by	Approved by	
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024		
Escala 2:1		Document type	Document status	
		Title M3 20mm	DWG No.	
			Rev.	Date of issue / Sheet 1/1

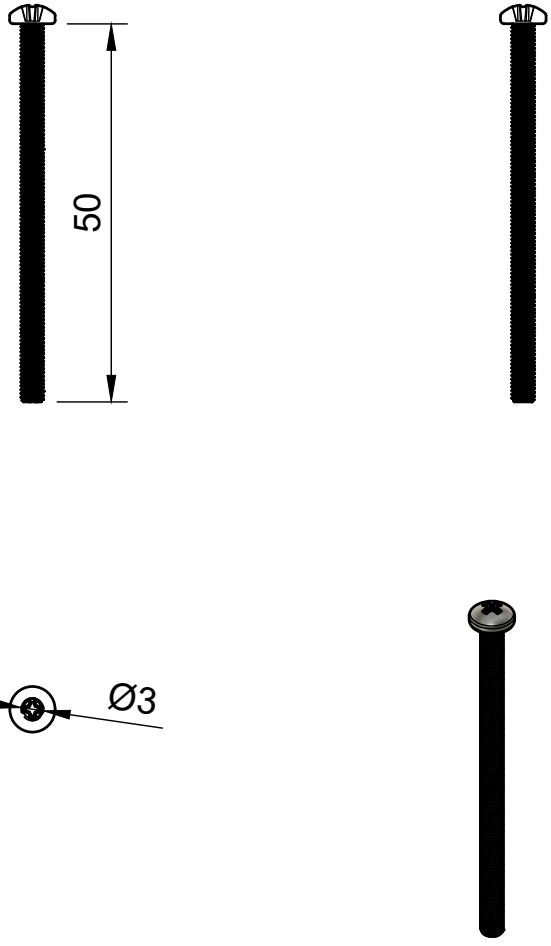




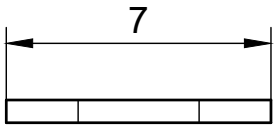


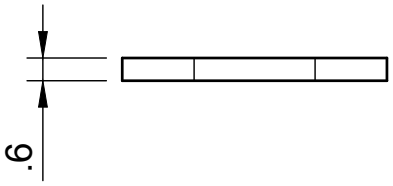


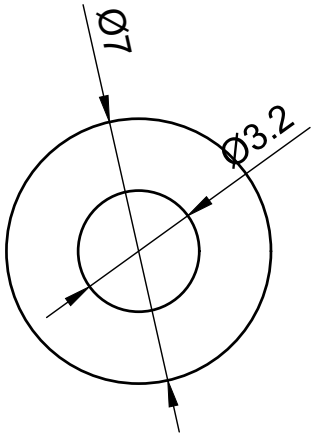
Dept.	Technical reference	Created by	Approved by	
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024		
Escala 2:1		Document type	Document status	
		Title M3 15mm	DWG No.	
			Rev.	Date of issue / Sheet 1/1

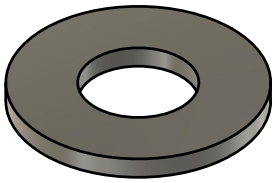


Dept.	Technical reference	Created by	Approved by	
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024		
Escala 1:1		Document type	Document status	
		Title M3 50mm	DWG No.	
			Rev.	Date of issue / Sheet 1/1

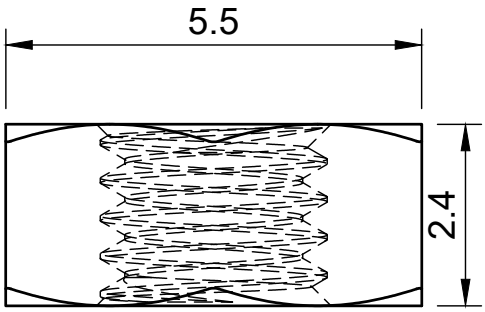


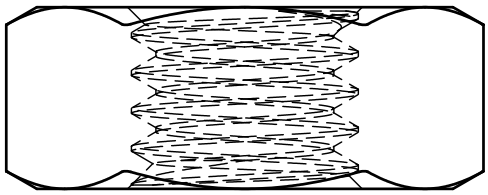


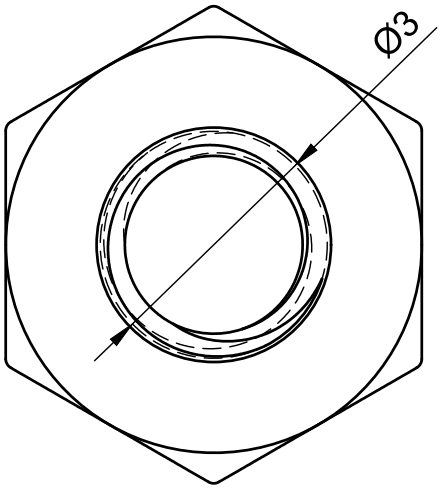


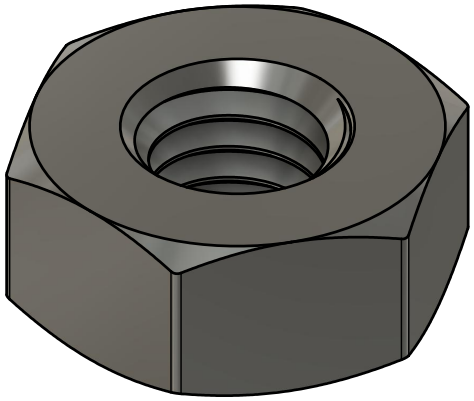


Dept.	Technical reference	Created by	Approved by	
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024		
Escala 5:1		Document type	Document status	
		Title Arandela M3	DWG No.	
			Rev. / Date of issue	Sheet 1/1









Dept.	Technical reference	Created by	Approved by	
	Dimensiones en mm	Fabrizio Rodríguez 23/06/2024		
Escala 10:1		Document type	Document status	
		Title Tuerca M3	DWG No.	
			Rev.	Date of issue / Sheet 1/1

10.8 Código fuente de los scripts

10.8.1 direct_kinematics.py

```python
import numpy as np

def calcular_posicion(theta1, theta2, theta3):
    l1 = 81.998
    l2 = 86.104
    l3 = 188.858

    def T(d, th, a, al):
        t = np.array([
            [np.cos(th), -np.cos(al) * np.sin(th),
             np.sin(al) * np.sin(th), a * np.cos(th)],
            [np.sin(th),  np.cos(al) * np.cos(th),
             -np.sin(al) * np.cos(th), a * np.sin(th)],
            [0, np.sin(al), np.cos(al), d],
            [0, 0, 0, 1],
        ])
        return t

    d = np.array([0, 0, 0])
    theta = np.array([theta1, theta2, theta3])
    a = np.array([l1, l2, l3])
    al = np.array([np.pi / 2, 0, 0])

    O00 = np.transpose([[0, 0, 0, 1]])
    O11 = np.transpose([[0, 0, 0, 1]])
    O22 = np.transpose([[0, 0, 0, 1]])
    O33 = np.transpose([[0, 0, 0, 1]])

    T01 = T(d[0], theta[0], a[0], al[0])
    T12 = T(d[1], theta[1], a[1], al[1])
    T23 = T(d[2], theta[2], a[2], al[2])

    T02 = np.dot(T01, T12)
    T03 = np.dot(T02, T23)

    O30 = np.dot(T03, O33)

    x3 = O30[0]
    y3 = O30[1]
    z3 = O30[2]

    pos = [x3, y3, z3]
    return pos
```

10.8.2 inverse_kinematics.py

```python
import math
import numpy as np

def calcular_angulos(x, y, z):
    # Dimensiones de la pata
    l1 = 61.197 # coxa
    l2 = 81.31   # fémur
    l3 = 155.398 # tibia
    codo = 1 # CODO ARRIBA = 0 / CODO ABAJO = 1

    theta1 = math.atan2(y, x)
    x = x - l1 * math.cos(theta1)
    y = y - l1 * math.sin(theta1)

    # Comprobación punto alcanzable
    c3 = (x**2 + y**2 + z**2 - l2**2 - l3**2)
    c3 /= (2 * l2 * l3)

    if abs(c3) > 1:
        print("El punto NO es alcanzable")
        return

    # Si el punto es alcanzable
    theta3 = -math.acos(c3)

    phi = math.atan2(z, math.sqrt(x**2 + y**2))
    alfa = math.atan2(l3 * math.sin(theta3), l2 +
                      l3 * math.cos(theta3))
    theta2 = phi - alfa

    thetas = np.array([theta1, theta2, theta3])

    return thetas
```

10.8.3 process_data.py

```
import matplotlib.pyplot as plt
import numpy as np

def grafica_q(thetas):
    thetas1 = [arr[0][0] for arr in thetas]
    thetas2 = [arr[0][1] for arr in thetas]
    thetas3 = [arr[0][2] for arr in thetas]
    t = [arr[1] for arr in thetas]

    thetas1p = np.diff(thetas1) / np.diff(t)
    thetas2p = np.diff(thetas2) / np.diff(t)
    thetas3p = np.diff(thetas3) / np.diff(t)

    thetas1pp = np.diff(thetas1p) / np.diff(t[:-1])
    thetas2pp = np.diff(thetas2p) / np.diff(t[:-1])
    thetas3pp = np.diff(thetas3p) / np.diff(t[:-1])

    fig, axs = plt.subplots(3, 3)
    axs[0, 0].plot(t, thetas1, marker='o')
    axs[0, 1].plot(t, thetas2, marker='o')
    axs[0, 2].plot(t, thetas3, marker='o')

    axs[1, 0].plot(t[:-1], thetas1p, marker='o')
    axs[1, 1].plot(t[:-1], thetas2p, marker='o')
    axs[1, 2].plot(t[:-1], thetas3p, marker='o')

    axs[2, 0].plot(t[:-2], thetas1pp, marker='o')
    axs[2, 1].plot(t[:-2], thetas2pp, marker='o')
    axs[2, 2].plot(t[:-2], thetas3pp, marker='o')

    axs[0, 0].set_title('Posición theta1')
    axs[0, 0].set_xlabel('Tiempo')
    axs[0, 0].set_ylabel('Theta1')
    axs[0, 1].set_title('Posición theta2')
    axs[0, 1].set_xlabel('Tiempo')
    axs[0, 1].set_ylabel('Theta2')
    axs[0, 2].set_title('Posición theta3')
    axs[0, 2].set_xlabel('Tiempo')
    axs[0, 2].set_ylabel('Theta3')
    axs[1, 0].set_title('Velocidad theta1')
    axs[1, 0].set_xlabel('Tiempo')
    axs[1, 0].set_ylabel('Theta1p')
    axs[1, 1].set_title('Velocidad theta2')
    axs[1, 1].set_xlabel('Tiempo')
    axs[1, 1].set_ylabel('Theta2p')
    axs[1, 2].set_title('Velocidad theta3')
    axs[1, 2].set_xlabel('Tiempo')
    axs[1, 2].set_ylabel('Theta3p')
    axs[2, 0].set_title('Aceleración theta1')
    axs[2, 0].set_xlabel('Tiempo')
    axs[2, 0].set_ylabel('Theta1pp')
    axs[2, 1].set_title('Aceleración theta2')
    axs[2, 1].set_xlabel('Tiempo')
    axs[2, 1].set_ylabel('Theta2pp')
    axs[2, 2].set_title('Aceleración theta3')
```

```
    axs[2, 2].set_xlabel('Tiempo')
    axs[2, 2].set_ylabel('Theta3pp')
    plt.show()

def grafica_p(pos):
    posx = [arr[0][0] for arr in pos]
    posy = [arr[0][1] for arr in pos]
    posz = [arr[0][2] for arr in pos]
    t = [arr[1] for arr in pos]

    fig, axs = plt.subplots(1, 3)

    axs[0].plot(t, posx, marker='o')
    axs[1].plot(t, posy, marker='o')
    axs[2].plot(t, posz, marker='o')
    plt.show()

    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    ax.scatter(posx, posy, posz, c='r', marker='o')

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')

    plt.axis('equal')

    plt.show()
```

10.8.4 trapezoidal_interpolation.py

```
import numpy as np

def calcular_qt(thetas0, thetasf , t, T):
    theta_t = np.array([0.001, 0.001, 0.001])

    # Valores para el interpolador
    difTh = thetasf - thetas0
    V = 1.9 * difTh / T

    # Inicializamos tau y a
    tau = np.zeros_like(V)
    a = np.zeros_like(V)

    # Usamos una máscara para encontrar los índices
    mask = V != 0

    # Calculamos tau y a
    tau[mask] = (thetas0[mask] - thetasf[mask] +
                 V[mask] * T) / V[mask]
    a[mask] = V[mask] / tau[mask]

    def interpolation(th0, thf, V, a, tau, t, T):
        if th0 == thf:
            return th0
        if t <= tau:
            theta_t = th0 + a / 2 * t**2
        if tau < t <= (T - tau):
            theta_t = (thf + th0 - V * T) / 2 + V * t
        if (T - tau) < t <= T:
            theta_t = (thf - a / 2 * T**2 + a * T * t -
                       a / 2 * t**2)
        return theta_t

    i = 0
    while i < 3:
        temp = interpolation(thetas0[i], thetasf[i],
                             V[i], a[i], tau[i], t, T)
        theta_t[i] = temp
        i += 1

    return theta_t
```

10.8.5 spline_cubico.py

```python
import numpy as np

def calcular_qt(puntos, tiempos, t):
    n = len(puntos) - 1
    h = np.diff(tiempos)

    # Definir matrices A y b
    A = np.zeros((n + 1, n + 1))
    b = np.zeros((n + 1, 3))

    # Condiciones de frontera
    A[0, 0] = 1
    A[-1, -1] = 1
    b[0] = [0, 0, 0]
    b[-1] = [0, 0, 0]

    for i in range(1, n):
        A[i, i - 1] = h[i - 1]
        A[i, i] = 2 * (h[i - 1] + h[i])
        A[i, i + 1] = h[i]
        b[i] = 3 * ((puntos[i + 1] - puntos[i]) / h[i] -
                    (puntos[i] - puntos[i - 1]) / h[i - 1])

    # Resolver el sistema de ecuaciones
    c = np.linalg.solve(A, b)

    a = puntos[:-1]
    b = np.zeros((n, 3))
    d = np.zeros((n, 3))

    for i in range(n):
        b[i] = ((puntos[i + 1] - puntos[i]) / h[i] -
                h[i] * (2 * c[i] + c[i + 1]) / 3)
        d[i] = (c[i + 1] - c[i]) / (3 * h[i])

    def spline(t, i):
        dt = t - tiempos[i]
        return (a[i] + b[i] * dt + c[i] * dt**2 +
                d[i] * dt**3)

    # Encontrar el intervalo
    for i in range(n):
        if tiempos[i] <= t <= tiempos[i + 1]:
            return spline(t, i)

    return None
```

10.8.6 Rotation.py

```python
# Definimos el radio de la circunferencia
# como 97.32 + 126 mm que es el punto
# de reposo del hexapod para las patas
# 2 y 5 medido desde el centro.
# Por tanto el Radio será: 223.32mm

# Para las patas 1, 3, 4 y 6 el centro
# de la circunferencia está desplazado:
# -115.004mm en X y 11.113mm en Y.
# Para las patas 2 y 5 el centro de la
# circunferencia está desplazado:
# -97.32mm en X y 0mm en Y.

# Por tanto se puede aplicar la siguiente
# ecuación: siendo h el desfase en X y
# k el desfase en Y
# r = 223.32mm
# x_points = h + r * np.cos(angles_radians)
# y_points = k + r * np.sin(angles_radians)

import numpy as np
import trapezoidal_interpolation as trapint
import inverse_kinematics as ink
import process_data as pdata

def semicircle_generatorxy(radio, altura_z,
                           puntos, T, hx, ky,
                           a0, af):
    p0 = np.array([a0, a0, a0]) # angulo inicial
    pf = np.array([af, af, af]) # angulo final
    t = 0 # iniciamos el tiempo a 0

    poses = []
    thetas = []
    # Generamos puntos a lo largo de la circunferencia
    # mediante un interpolador trapezoidal.
    for t in np.linspace(0, T, puntos):
        angulos = trapint.calcular_qt(p0, pf, t, T)
        # hacemos la correspondencia angulo
        # --> punto del espacio para la pata
        x = hx + radio * np.cos(angulos[0])
        y = ky + radio * np.sin(angulos[0])
        z = altura_z

        pos_t = np.array([x, y, z])
        poses.append((pos_t, t))
        theta_t = ink.calcular_angulos(*pos_t)
        thetas.append((theta_t, t))

    return poses, thetas
```

10.8.7 Rotation parametros us.py

```
import inverse_kinematics \
    as ink
import process_data as pdata
import numpy as np
import Rotation
import spline_cubico as sc

# Parametros comunes
T = 2 # s tiempo final
Treposo = 1 # s tiempo final
tiemposaire = np.array(
    [0, T/2, T])
tiemposairereposo = np.array(
    [0, Treposo/2, Treposo])
puntos = T * 105
puntosreposo = Treposo * 105
radio = 223.32
altura_z = -114

# ROTACION ARRASTRES
# ROTACION ARRASTRES PATA 2
ticks_Rotarrastre2 = []
hx = -97.32
ky = 0
a0 = 15.3 * (np.pi / 180)
af = -15.3 * (np.pi / 180)
poses, Thetas_Rotarrastre2 = \
    Rotation.semicircle_generatorxy(
        radio, altura_z, puntos, T,
        hx, ky, a0, af)
p025 = poses[-1]
pm = np.array([126, 0, altura_z])
pf25 = poses[0]

protairep25 = np.array([
    [p025[0][0], p025[0][1],
     p025[0][2]],
    [pm[0], pm[1], -80],
    [pf25[0][0], pf25[0][1],
     pf25[0][2]]
])
for theta_t, t in Thetas_Rotarrastre2:
    angulo_t = np.array([
        ((theta_t[0] * 180.0) / np.pi)
        + 90,
        ((theta_t[1] * 180.0) / np.pi)
        + 90,
        360 - ((theta_t[2] * 180.0)
        / np.pi)
    ])
    if angulo_t[2] < 0:
        angulo_t[2] += 360
    if angulo_t[2] > 360:
        angulo_t[2] -= 360
    us = np.array([
```

```
56              (205.0 / 18.0) * angulo_t[0]
57              + 1069.0 / 2.0,
58              (313.0 / 30.0) * angulo_t[1]
59              + 1187.0 / 2.0,
60              (199.0 / 18.0) * angulo_t[2]
61              + 1005.0 / 2.0
62          ])
63          ticks_t = np.array([
64              int(us[0] / 4.88),
65              int(us[1] / 4.88),
66              int(us[2] / 4.88)
67          ])
68          ticks_Rotarrastre2.append(
69              (us.astype(int), t))
70
71      # ROTACION ARRASTRES PATA 5
72      ticks_Rotarrastre5 = []
73      hx = -97.32
74      ky = 0
75      a0 = 15.3 * (np.pi / 180)
76      af = -15.3 * (np.pi / 180)
77      poses, Thetas_Rotarrastre5 = \
78          Rotation.semicircle_generatorxy(
79              radio, altura_z, puntos, T,
80              hx, ky, a0, af)
81      p025 = poses[-1]
82      pm = np.array([126, 0, altura_z])
83      pf25 = poses[0]
84      for theta_t, t in Thetas_Rotarrastre5:
85          angulo_t = np.array([
86              ((theta_t[0] * 180.0) / np.pi)
87              + 90,
88              ((theta_t[1] * 180.0) / np.pi)
89              + 90,
90              360 - ((theta_t[2] * 180.0)
91              / np.pi)
92          ])
93          if angulo_t[2] < 0:
94              angulo_t[2] += 360
95          if angulo_t[2] > 360:
96              angulo_t[2] -= 360
97          us = np.array([
98              (361.0 / 30.0) * angulo_t[0]
99              + 661.0 / 2.0,
100             (97.0 / 9.0) * angulo_t[1]
101             + 550.0,
102             (524.0 / 45.0) * angulo_t[2]
103             + 523.0
104         ])
105         ticks_t = np.array([
106             int(us[0] / 4.88),
107             int(us[1] / 4.88),
108             int(us[2] / 4.88)
109         ])
110         ticks_Rotarrastre5.append(
111             (us.astype(int), t))
112
```

```
113 # ROTACION ARRASTRES PATA 1
114 ticks_Rotarrastre1 = []
115 hx = -115.004
116 ky = -11.113
117 a0 = 13.15 * (np.pi / 180)
118 af = -7.44 * (np.pi / 180)
119 poses, Thetas_Rotarrastre1 = \
120     Rotation.semicircle_generatorxy(
121         radio, altura_z, puntos, T,
122         hx, ky, a0, af)
123 p014 = poses[-1]
124 pm1346 = np.array([108.039, 0,
125                    altura_z])
126 pf14 = poses[0]
127 protairep14 = np.array([
128     [p014[0][0], p014[0][1],
129      p014[0][2]],
130     [pm1346[0], pm1346[1],
131      -80],
132     [pf14[0][0], pf14[0][1],
133      pf14[0][2]]
134 ])
135 for theta_t, t in Thetas_Rotarrastre1:
136     angulo_t = np.array([
137         ((theta_t[0] * 180.0) / np.pi)
138         + 90,
139         ((theta_t[1] * 180.0) / np.pi)
140         + 90,
141         360 - ((theta_t[2] * 180.0)
142         / np.pi)
143     ])
144     if angulo_t[2] < 0:
145         angulo_t[2] += 360
146     if angulo_t[2] > 360:
147         angulo_t[2] -= 360
148     us = np.array([
149         (983.0 / 90.0) * angulo_t[0]
150         + 863.0 / 2.0,
151         (989.0 / 90.0) * angulo_t[1]
152         + 937.0 / 2.0,
153         (1079.0 / 90.0) * angulo_t[2]
154         + 683.0 / 2.0
155     ])
156     ticks_t = np.array([
157         int(us[0] / 4.88),
158         int(us[1] / 4.88),
159         int(us[2] / 4.88)
160     ])
161     ticks_Rotarrastre1.append(
162         (us.astype(int), t))
163
164 # ROTACION ARRASTRES PATA 4
165 ticks_Rotarrastre4 = []
166 hx = -115.004
167 ky = -11.113
168 a0 = 18.15 * (np.pi / 180)
169 af = -12.44 * (np.pi / 180)
```

```
poses, Thetas_Rotarrastre4 = \
    Rotation.semicircle_generatorxy(
        radio, altura_z, puntos, T,
        hx, ky, a0, af)
p014 = poses[-1]
pm1346 = np.array([108.039, 0,
                   altura_z])
pf14 = poses[0]
for theta_t, t in Thetas_Rotarrastre4:
    angulo_t = np.array([
        ((theta_t[0] * 180.0) / np.pi)
        + 90,
        ((theta_t[1] * 180.0) / np.pi)
        + 90,
        360 - ((theta_t[2] * 180.0)
        / np.pi)
    ])
    if angulo_t[2] < 0:
        angulo_t[2] += 360
    if angulo_t[2] > 360:
        angulo_t[2] -= 360
    us = np.array([
        (502.0 / 45.0) * angulo_t[0]
        + 464,
        (989.0 / 90.0) * angulo_t[1]
        + 901.0 / 2.0,
        (98.0 / 9.0) * angulo_t[2]
        + 468.0
    ])
    ticks_t = np.array([
        int(us[0] / 4.88),
        int(us[1] / 4.88),
        int(us[2] / 4.88)
    ])
    ticks_Rotarrastre4.append(
        (us.astype(int), t))

# ROTACION ARRASTRES PATA 3
ticks_Rotarrastre3 = []
hx = -115.004
ky = 11.113
a0 = 12.44 * (np.pi / 180)
af = -18.15 * (np.pi / 180)
poses, Thetas_Rotarrastre3 = \
    Rotation.semicircle_generatorxy(
        radio, altura_z, puntos, T,
        hx, ky, a0, af)
p036 = poses[-1]
pf36 = poses[0]
protairep36 = np.array([
    [p036[0][0], p036[0][1],
     p036[0][2]],
    [pm1346[0], pm1346[1],
     -80],
    [pf36[0][0], pf36[0][1],
     pf36[0][2]]
])
```

```
227 for theta_t, t in Thetas_Rotarrastre3:
228     angulo_t = np.array([
229         ((theta_t[0] * 180.0) / np.pi)
230         + 90,
231         ((theta_t[1] * 180.0) / np.pi)
232         + 90,
233         360 - ((theta_t[2] * 180.0)
234         / np.pi)
235     ])
236     if angulo_t[2] < 0:
237         angulo_t[2] += 360
238     if angulo_t[2] > 360:
239         angulo_t[2] -= 360
240     us = np.array([
241         (496.0 / 45.0) * angulo_t[0]
242         + 514.0,
243         (959.0 / 90.0) * angulo_t[1]
244         + 1029.0 / 2.0,
245         (491.0 / 45.0) * angulo_t[2]
246         + 507
247     ])
248     ticks_t = np.array([
249         int(us[0] / 4.88),
250         int(us[1] / 4.88),
251         int(us[2] / 4.88)
252     ])
253     ticks_Rotarrastre3.append(
254         (us.astype(int), t))
255
256 # ROTACION ARRASTRES PATA 6
257 ticks_Rotarrastre6 = []
258 hx = -115.004
259 ky = 11.113
260 a0 = 12.44 * (np.pi / 180)
261 af = -18.15 * (np.pi / 180)
262 poses, Thetas_Rotarrastre6 = \
263     Rotation.semicircle_generatorxy(
264         radio, altura_z, puntos, T,
265         hx, ky, a0, af)
266 p036 = poses[-1]
267 pf36 = poses[0]
268 for theta_t, t in Thetas_Rotarrastre6:
269     angulo_t = np.array([
270         ((theta_t[0] * 180.0) / np.pi)
271         + 90,
272         ((theta_t[1] * 180.0) / np.pi)
273         + 90,
274         360 - ((theta_t[2] * 180.0)
275         / np.pi)
276     ])
277     if angulo_t[2] < 0:
278         angulo_t[2] += 360
279     if angulo_t[2] > 360:
280         angulo_t[2] -= 360
281     us = np.array([
282         (1021.0 / 90.0) * angulo_t[0]
283         + 1071.0 / 2.0,
```

```
284             (169.0 / 15.0) * angulo_t[1]
285             + 490.0,
286             (496.0 / 45.0) * angulo_t[2]
287             + 439.0
288         ])
289         ticks_t = np.array([
290             int(us[0] / 4.88),
291             int(us[1] / 4.88),
292             int(us[2] / 4.88)
293         ])
294         ticks_Rotarrastre6.append(
295             (us.astype(int), t))
296 
297     # ROTACION AIRES
298     # ROTACION AIRES PATA 2
299     poses = []
300     ticks_Rotaire2 = []
301     for t in np.linspace(
302             tiemposaire[0], tiemposaire[-1],
303             num=int((tiemposaire[-1] -
304             tiemposaire[0]) * 105)):
305         pos_t = sc.calcular_qt(
306             protairep25, tiemposaire, t)
307         poses.append((pos_t, t))
308         theta_t = ink.calcular_angulos(
309             *pos_t)
310         angulo_t = np.array([
311             ((theta_t[0] * 180.0) / np.pi)
312             + 90,
313             ((theta_t[1] * 180.0) / np.pi)
314             + 90,
315             360 - ((theta_t[2] * 180.0)
316             / np.pi)
317         ])
318         if angulo_t[2] < 0:
319             angulo_t[2] += 360
320         if angulo_t[2] > 360:
321             angulo_t[2] -= 360
322         us = np.array([
323             (205.0 / 18.0) * angulo_t[0]
324             + 1069.0 / 2.0,
325             (313.0 / 30.0) * angulo_t[1]
326             + 1187.0 / 2.0,
327             (199.0 / 18.0) * angulo_t[2]
328             + 1005.0 / 2.0
329         ])
330         ticks_t = np.array([
331             int(us[0] / 4.88),
332             int(us[1] / 4.88),
333             int(us[2] / 4.88)
334         ])
335         ticks_Rotaire2.append(
336             (us.astype(int), t))
337 
338     # ROTACION AIRES PATA 5
339     poses = []
340     ticks_Rotaire5 = []
```

```
for t in np.linspace(
        tiemposaire[0], tiemposaire[-1],
        num=int((tiemposaire[-1] -
        tiemposaire[0]) * 105)):
    pos_t = sc.calcular_qt(
        protairep25, tiemposaire, t)
    poses.append((pos_t, t))
    theta_t = ink.calcular_angulos(
        *pos_t)
    angulo_t = np.array([
        ((theta_t[0] * 180.0) / np.pi)
        + 90,
        ((theta_t[1] * 180.0) / np.pi)
        + 90,
        360 - ((theta_t[2] * 180.0)
        / np.pi)
    ])
    if angulo_t[2] < 0:
        angulo_t[2] += 360
    if angulo_t[2] > 360:
        angulo_t[2] -= 360
    us = np.array([
        (361.0 / 30.0) * angulo_t[0]
        + 661.0 / 2.0,
        (97.0 / 9.0) * angulo_t[1]
        + 550.0,
        (524.0 / 45.0) * angulo_t[2]
        + 523.0
    ])
    ticks_t = np.array([
        int(us[0] / 4.88),
        int(us[1] / 4.88),
        int(us[2] / 4.88)
    ])
    ticks_Rotaire5.append(
        (us.astype(int), t))

# ROTACION AIRES PATA 1
poses = []
ticks_Rotaire1 = []
for t in np.linspace(
        tiemposaire[0], tiemposaire[-1],
        num=int((tiemposaire[-1] -
        tiemposaire[0]) * 105)):
    pos_t = sc.calcular_qt(
        protairep14, tiemposaire, t)
    poses.append((pos_t, t))
    theta_t = ink.calcular_angulos(
        *pos_t)
    angulo_t = np.array([
        ((theta_t[0] * 180.0) / np.pi)
        + 90,
        ((theta_t[1] * 180.0) / np.pi)
        + 90,
        360 - ((theta_t[2] * 180.0)
        / np.pi)
    ])
```

```
398     if angulo_t[2] < 0:
399         angulo_t[2] += 360
400     if angulo_t[2] > 360:
401         angulo_t[2] -= 360
402     us = np.array([
403         (983.0 / 90.0) * angulo_t[0]
404         + 863.0 / 2.0,
405         (989.0 / 90.0) * angulo_t[1]
406         + 937.0 / 2.0,
407         (1079.0 / 90.0) * angulo_t[2]
408         + 683.0 / 2.0
409     ])
410     ticks_t = np.array([
411         int(us[0] / 4.88),
412         int(us[1] / 4.88),
413         int(us[2] / 4.88)
414     ])
415     ticks_Rotaire1.append(
416         (us.astype(int), t))
417 
418 # ROTACION AIRES PATA 4
419 poses = []
420 ticks_Rotaire4 = []
421 for t in np.linspace(
422         tiemposaire[0], tiemposaire[-1],
423         num=int((tiemposaire[-1] -
424         tiemposaire[0]) * 105)):
425     pos_t = sc.calcular_qt(
426         protairep14, tiemposaire, t)
427     poses.append((pos_t, t))
428     theta_t = ink.calcular_angulos(
429         *pos_t)
430     angulo_t = np.array([
431         ((theta_t[0] * 180.0) / np.pi)
432         + 90,
433         ((theta_t[1] * 180.0) / np.pi)
434         + 90,
435         360 - ((theta_t[2] * 180.0)
436         / np.pi)
437     ])
438     if angulo_t[2] < 0:
439         angulo_t[2] += 360
440     if angulo_t[2] > 360:
441         angulo_t[2] -= 360
442     us = np.array([
443         (502.0 / 45.0) * angulo_t[0]
444         + 464,
445         (989.0 / 90.0) * angulo_t[1]
446         + 901.0 / 2.0,
447         (98.0 / 9.0) * angulo_t[2]
448         + 468.0
449     ])
450     ticks_t = np.array([
451         int(us[0] / 4.88),
452         int(us[1] / 4.88),
453         int(us[2] / 4.88)
454     ])
```

```
455     ticks_Rotaire4.append(
456         (ticks_t, t))
457 
458 # ROTACION AIRES PATA 3
459 poses = []
460 ticks_Rotaire3 = []
461 for t in np.linspace(
462         tiemposaire[0], tiemposaire[-1],
463         num=int((tiemposaire[-1] -
464         tiemposaire[0]) * 105)):
465     pos_t = sc.calcular_qt(
466         protairep36, tiemposaire, t)
467     poses.append((pos_t, t))
468     theta_t = ink.calcular_angulos(
469         *pos_t)
470     angulo_t = np.array([
471         ((theta_t[0] * 180.0) / np.pi)
472         + 90,
473         ((theta_t[1] * 180.0) / np.pi)
474         + 90,
475         360 - ((theta_t[2] * 180.0)
476         / np.pi)
477     ])
478     if angulo_t[2] < 0:
479         angulo_t[2] += 360
480     if angulo_t[2] > 360:
481         angulo_t[2] -= 360
482     us = np.array([
483         (496.0 / 45.0) * angulo_t[0]
484         + 514.0,
485         (959.0 / 90.0) * angulo_t[1]
486         + 1029.0 / 2.0,
487         (491.0 / 45.0) * angulo_t[2]
488         + 507
489     ])
490     ticks_t = np.array([
491         int(us[0] / 4.88),
492         int(us[1] / 4.88),
493         int(us[2] / 4.88)
494     ])
495     ticks_Rotaire3.append(
496         (us.astype(int), t))
497 
498 # ROTACION AIRES PATA 6
499 poses = []
500 ticks_Rotaire6 = []
501 for t in np.linspace(
502         tiemposaire[0], tiemposaire[-1],
503         num=int((tiemposaire[-1] -
504         tiemposaire[0]) * 105)):
505     pos_t = sc.calcular_qt(
506         protairep36, tiemposaire, t)
507     poses.append((pos_t, t))
508     theta_t = ink.calcular_angulos(
509         *pos_t)
510     angulo_t = np.array([
511         ((theta_t[0] * 180.0) / np.pi)
```

```
512             + 90,
513             ((theta_t[1] * 180.0) / np.pi)
514             + 90,
515             360 - ((theta_t[2] * 180.0)
516             / np.pi)
517         ])
518         if angulo_t[2] < 0:
519             angulo_t[2] += 360
520         if angulo_t[2] > 360:
521             angulo_t[2] -= 360
522         us = np.array([
523             (1021.0 / 90.0) * angulo_t[0]
524             + 1071.0 / 2.0,
525             (169.0 / 15.0) * angulo_t[1]
526             + 490.0,
527             (496.0 / 45.0) * angulo_t[2]
528             + 439.0
529         ])
530         ticks_t = np.array([
531             int(us[0] / 4.88),
532             int(us[1] / 4.88),
533             int(us[2] / 4.88)
534         ])
535         ticks_Rotaire6.append(
536             (us.astype(int), t))
537
538     # TICKS DESDE REPOSO ROTACION
539     # AIRES DESDE REPOSO ROTACION 1
540     protairep1reposo = np.array([
541         [pm[0], pm[1], pm[2]],
542         [(pf14[0][0]+pm[0])/2,
543          (pf14[0][1]+pm[1])/2,
544          -90],
545         [pf14[0][0], pf14[0][1],
546          pf14[0][2]]
547     ])
548     poses = []
549     ticks_Rotaire1reposo = []
550     for t in np.linspace(
551             tiemposairereposo[0],
552             tiemposairereposo[-1],
553             num=int((tiemposairereposo[-1] -
554             tiemposairereposo[0]) * 105)):
555         pos_t = sc.calcular_qt(
556             protairep1reposo,
557             tiemposairereposo, t)
558         poses.append((pos_t, t))
559         theta_t = ink.calcular_angulos(
560             *pos_t)
561         angulo_t = np.array([
562             ((theta_t[0] * 180.0) / np.pi)
563             + 90,
564             ((theta_t[1] * 180.0) / np.pi)
565             + 90,
566             360 - ((theta_t[2] * 180.0)
567             / np.pi)
568         ])
```

```
569     if angulo_t[2] < 0:
570         angulo_t[2] += 360
571     if angulo_t[2] > 360:
572         angulo_t[2] -= 360
573     us = np.array([
574         (983.0 / 90.0) * angulo_t[0]
575         + 863.0 / 2.0,
576         (989.0 / 90.0) * angulo_t[1]
577         + 937.0 / 2.0,
578         (1079.0 / 90.0) * angulo_t[2]
579         + 683.0 / 2.0
580     ])
581     ticks_t = np.array([
582         int(us[0] / 4.88),
583         int(us[1] / 4.88),
584         int(us[2] / 4.88)
585     ])
586     ticks_Rotaire1reposo.append(
587         (us.astype(int), t))
588 
589 # AIRES DESDE REPOSO ROTACION 3
590 protairep3reposo = np.array([
591     [pm[0], pm[1], pm[2]],
592     [(pf36[0][0]+pm[0])/2,
593      (pf36[0][1]+pm[1])/2,
594      -90],
595     [pf36[0][0], pf36[0][1],
596      pf36[0][2]]
597 ])
598 poses = []
599 ticks_Rotaire3reposo = []
600 for t in np.linspace(
601         tiemposairereposo[0],
602         tiemposairereposo[-1],
603         num=int((tiemposairereposo[-1] -
604         tiemposairereposo[0]) * 105)):
605     pos_t = sc.calcular_qt(
606         protairep3reposo,
607         tiemposairereposo, t)
608     poses.append((pos_t, t))
609     theta_t = ink.calcular_angulos(
610         *pos_t)
611     angulo_t = np.array([
612         ((theta_t[0] * 180.0) / np.pi)
613         + 90,
614         ((theta_t[1] * 180.0) / np.pi)
615         + 90,
616         360 - ((theta_t[2] * 180.0)
617         / np.pi)
618     ])
619     if angulo_t[2] < 0:
620         angulo_t[2] += 360
621     if angulo_t[2] > 360:
622         angulo_t[2] -= 360
623     us = np.array([
624         (496.0 / 45.0) * angulo_t[0]
625         + 514.0,
```

```
626             (959.0 / 90.0) * angulo_t[1]
627             + 1029.0 / 2.0,
628             (491.0 / 45.0) * angulo_t[2]
629             + 507
630         ])
631         ticks_t = np.array([
632             int(us[0] / 4.88),
633             int(us[1] / 4.88),
634             int(us[2] / 4.88)
635         ])
636         ticks_Rotaire3reposo.append(
637             (us.astype(int), t))
638 
639 # AIRES DESDE REPOSO ROTACION 5
640 protairep5reposo = np.array([
641     [pm[0], pm[1], pm[2]],
642     [(pf25[0][0]+pm[0])/2,
643      (pf25[0][1]+pm[1])/2,
644      -90],
645     [pf25[0][0], pf25[0][1],
646      pf25[0][2]]
647 ])
648 poses = []
649 ticks_Rotaire5reposo = []
650 for t in np.linspace(
651             tiemposairereposo[0],
652             tiemposairereposo[-1],
653             num=int((tiemposairereposo[-1] -
654             tiemposairereposo[0]) * 105)):
655         pos_t = sc.calcular_qt(
656             protairep5reposo,
657             tiemposairereposo, t)
658         poses.append((pos_t, t))
659         theta_t = ink.calcular_angulos(
660             *pos_t)
661         angulo_t = np.array([
662             ((theta_t[0] * 180.0) / np.pi)
663             + 90,
664             ((theta_t[1] * 180.0) / np.pi)
665             + 90,
666             360 - ((theta_t[2] * 180.0)
667             / np.pi)
668         ])
669         if angulo_t[2] < 0:
670             angulo_t[2] += 360
671         if angulo_t[2] > 360:
672             angulo_t[2] -= 360
673         us = np.array([
674             (361.0 / 30.0) * angulo_t[0]
675             + 661.0 / 2.0,
676             (97.0 / 9.0) * angulo_t[1]
677             + 550.0,
678             (524.0 / 45.0) * angulo_t[2]
679             + 523.0
680         ])
681         ticks_t = np.array([
682             int(us[0] / 4.88),
```

```
683             int(us[1] / 4.88),
684             int(us[2] / 4.88)
685         ])
686         ticks_Rotaire5reposo.append(
687             (us.astype(int), t))
688 
689 # ARRASTRE DESDE REPOSO ROTACION 4
690 ticks_Rotarrastre4reposo = []
691 hx = -115.004
692 ky = -11.113
693 a0 = 0 * (np.pi / 180)
694 af = -12.44 * (np.pi / 180)
695 poses, Thetas_Rotarrastre4reposo = \
696     Rotation.semicircle_generatorxy(
697         radio, altura_z,
698         puntosreposo, Treposo,
699         hx, ky, a0, af)
700 for theta_t, t in
701 Thetas_Rotarrastre4reposo:
702     angulo_t = np.array([
703         ((theta_t[0] * 180.0) / np.pi)
704         + 90,
705         ((theta_t[1] * 180.0) / np.pi)
706         + 90,
707         360 - ((theta_t[2] * 180.0)
708         / np.pi)
709     ])
710     if angulo_t[2] < 0:
711         angulo_t[2] += 360
712     if angulo_t[2] > 360:
713         angulo_t[2] -= 360
714     us = np.array([
715         (502.0 / 45.0) * angulo_t[0]
716         + 464,
717         (989.0 / 90.0) * angulo_t[1]
718         + 901.0 / 2.0,
719         (98.0 / 9.0) * angulo_t[2]
720         + 468.0
721     ])
722     ticks_t = np.array([
723         int(us[0] / 4.88),
724         int(us[1] / 4.88),
725         int(us[2] / 4.88)
726     ])
727     ticks_Rotarrastre4reposo.append(
728         (us.astype(int), t))
729 
730 # ARRASTRE DESDE REPOSO ROTACION 6
731 ticks_Rotarrastre6reposo = []
732 hx = -115.004
733 ky = 11.113
734 a0 = 0 * (np.pi / 180)
735 af = -18.15 * (np.pi / 180)
736 poses, Thetas_Rotarrastre6reposo = \
737     Rotation.semicircle_generatorxy(
738         radio, altura_z,
739         puntosreposo, Treposo,
```

```
740         hx, ky, a0, af)
741 for theta_t, t in
742 Thetas_Rotarrastre6reposo:
743     angulo_t = np.array([
744         ((theta_t[0] * 180.0) / np.pi)
745         + 90,
746         ((theta_t[1] * 180.0) / np.pi)
747         + 90,
748         360 - ((theta_t[2] * 180.0)
749         / np.pi)
750     ])
751     if angulo_t[2] < 0:
752         angulo_t[2] += 360
753     if angulo_t[2] > 360:
754         angulo_t[2] -= 360
755     us = np.array([
756         (1021.0 / 90.0) * angulo_t[0]
757         + 1071.0 / 2.0,
758         (169.0 / 15.0) * angulo_t[1]
759         + 490.0,
760         (496.0 / 45.0) * angulo_t[2]
761         + 439.0
762     ])
763     ticks_t = np.array([
764         int(us[0] / 4.88),
765         int(us[1] / 4.88),
766         int(us[2] / 4.88)
767     ])
768     ticks_Rotarrastre6reposo.append(
769         (us.astype(int), t))
770
771 # ARRASTRE DESDE REPOSO ROTACION 2
772 ticks_Rotarrastre2reposo = []
773 hx = -97.32
774 ky = 0
775 a0 = 0 * (np.pi / 180)
776 af = -15.3 * (np.pi / 180)
777 poses, Thetas_Rotarrastre2reposo = \
778     Rotation.semicircle_generatorxy(
779         radio, altura_z,
780         puntosreposo, Treposo,
781         hx, ky, a0, af)
782 for theta_t, t in
783 Thetas_Rotarrastre2reposo:
784     angulo_t = np.array([
785         ((theta_t[0] * 180.0) / np.pi)
786         + 90,
787         ((theta_t[1] * 180.0) / np.pi)
788         + 90,
789         360 - ((theta_t[2] * 180.0)
790         / np.pi)
791     ])
792     if angulo_t[2] < 0:
793         angulo_t[2] += 360
794     if angulo_t[2] > 360:
795         angulo_t[2] -= 360
796     us = np.array([
```

```
797             (205.0 / 18.0) * angulo_t[0]
798             + 1069.0 / 2.0,
799             (313.0 / 30.0) * angulo_t[1]
800             + 1187.0 / 2.0,
801             (199.0 / 18.0) * angulo_t[2]
802             + 1005.0 / 2.0
803         ])
804         ticks_t = np.array([
805             int(us[0] / 4.88),
806             int(us[1] / 4.88),
807             int(us[2] / 4.88)
808         ])
809         ticks_Rotarrastre2reposo.append(
810             (us.astype(int), t))
811
812 # TICKS DESDE REPOSO RETURN
813 # ARRASTRE DE RETURN ROTACION 1
814 ticks_Rotarrastre1return = []
815 hx = -115.004
816 ky = -11.113
817 a0 = 13.15 * (np.pi / 180)
818 af = 0 * (np.pi / 180)
819 poses, Thetas_Rotarrastre1return = \
820     Rotation.semicircle_generatorxy(
821         radio, altura_z,
822         puntosreposo, Treposo,
823         hx, ky, a0, af)
824 for theta_t, t in
825 Thetas_Rotarrastre1return:
826     angulo_t = np.array([
827         ((theta_t[0] * 180.0) / np.pi)
828         + 90,
829         ((theta_t[1] * 180.0) / np.pi)
830         + 90,
831         360 - ((theta_t[2] * 180.0)
832         / np.pi)
833     ])
834     if angulo_t[2] < 0:
835         angulo_t[2] += 360
836     if angulo_t[2] > 360:
837         angulo_t[2] -= 360
838     us = np.array([
839         (983.0 / 90.0) * angulo_t[0]
840         + 863.0 / 2.0,
841         (989.0 / 90.0) * angulo_t[1]
842         + 937.0 / 2.0,
843         (1079.0 / 90.0) * angulo_t[2]
844         + 683.0 / 2.0
845     ])
846     ticks_t = np.array([
847         int(us[0] / 4.88),
848         int(us[1] / 4.88),
849         int(us[2] / 4.88)
850     ])
851     ticks_Rotarrastre1return.append(
852         (us.astype(int), t))
853
```

```
854 # ARRASTRE DE RETURN ROTACION 3
855 ticks_Rotarrastre3return = []
856 hx = -115.004
857 ky = 11.113
858 a0 = 12.44 * (np.pi / 180)
859 af = 0 * (np.pi / 180)
860 poses, Thetas_Rotarrastre3return = \
861     Rotation.semicircle_generatorxy(
862         radio, altura_z,
863         puntosreposo, Treposo,
864         hx, ky, a0, af)
865 for theta_t, t in
866 Thetas_Rotarrastre3return:
867     angulo_t = np.array([
868         ((theta_t[0] * 180.0) / np.pi)
869         + 90,
870         ((theta_t[1] * 180.0) / np.pi)
871         + 90,
872         360 - ((theta_t[2] * 180.0)
873         / np.pi)
874     ])
875     if angulo_t[2] < 0:
876         angulo_t[2] += 360
877     if angulo_t[2] > 360:
878         angulo_t[2] -= 360
879     us = np.array([
880         (496.0 / 45.0) * angulo_t[0]
881         + 514.0,
882         (959.0 / 90.0) * angulo_t[1]
883         + 1029.0 / 2.0,
884         (491.0 / 45.0) * angulo_t[2]
885         + 507
886     ])
887     ticks_t = np.array([
888         int(us[0] / 4.88),
889         int(us[1] / 4.88),
890         int(us[2] / 4.88)
891     ])
892     ticks_Rotarrastre3return.append(
893         (us.astype(int), t))
894
895 # ARRASTRE DE RETURN ROTACION 5
896 ticks_Rotarrastre5return = []
897 hx = -97.32
898 ky = 0
899 a0 = 15.3 * (np.pi / 180)
900 af = 0 * (np.pi / 180)
901 poses, Thetas_Rotarrastre5return = \
902     Rotation.semicircle_generatorxy(
903         radio, altura_z,
904         puntosreposo, Treposo,
905         hx, ky, a0, af)
906 for theta_t, t in
907 Thetas_Rotarrastre5return:
908     angulo_t = np.array([
909         ((theta_t[0] * 180.0) / np.pi)
910         + 90,
```

```
911             ((theta_t[1] * 180.0) / np.pi)
912             + 90,
913             360 - ((theta_t[2] * 180.0)
914             / np.pi)
915         ])
916         if angulo_t[2] < 0:
917             angulo_t[2] += 360
918         if angulo_t[2] > 360:
919             angulo_t[2] -= 360
920         us = np.array([
921             (361.0 / 30.0) * angulo_t[0]
922             + 661.0 / 2.0,
923             (97.0 / 9.0) * angulo_t[1]
924             + 550.0,
925             (524.0 / 45.0) * angulo_t[2]
926             + 523.0
927         ])
928         ticks_t = np.array([
929             int(us[0] / 4.88),
930             int(us[1] / 4.88),
931             int(us[2] / 4.88)
932         ])
933         ticks_Rotarrastre5return.append(
934             (us.astype(int), t))
935 
936 # AIRE DE RETURN ROTACION 2
937 poses = []
938 ticks_Rotaire2return = []
939 protairep2return = np.array([
940     [p025[0][0], p025[0][1],
941      p025[0][2]],
942     [(p025[0][0]+pm[0])/2,
943      (p025[0][1]+pm[1])/2,
944      -90],
945     [pm[0], pm[1], pm[2]]
946 ])
947 for t in np.linspace(
948             tiemposairereposo[0],
949             tiemposairereposo[-1],
950             num=int((tiemposairereposo[-1] -
951             tiemposairereposo[0]) * 105)):
952         pos_t = sc.calcular_qt(
953             protairep2return,
954             tiemposairereposo, t)
955         poses.append((pos_t, t))
956         theta_t = ink.calcular_angulos(
957             *pos_t)
958         angulo_t = np.array([
959             ((theta_t[0] * 180.0) / np.pi)
960             + 90,
961             ((theta_t[1] * 180.0) / np.pi)
962             + 90,
963             360 - ((theta_t[2] * 180.0)
964             / np.pi)
965         ])
966         if angulo_t[2] < 0:
967             angulo_t[2] += 360
```

```
968      if angulo_t[2] > 360:
969          angulo_t[2] -= 360
970      us = np.array([
971          (205.0 / 18.0) * angulo_t[0]
972          + 1069.0 / 2.0,
973          (313.0 / 30.0) * angulo_t[1]
974          + 1187.0 / 2.0,
975          (199.0 / 18.0) * angulo_t[2]
976          + 1005.0 / 2.0
977      ])
978      ticks_t = np.array([
979          int(us[0] / 4.88),
980          int(us[1] / 4.88),
981          int(us[2] / 4.88)
982      ])
983      ticks_Rotaire2return.append(
984          (us.astype(int), t))
985
986  # AIRE DE RETURN ROTACION 4
987  poses = []
988  ticks_Rotaire4return = []
989  protairep4return = np.array([
990      [p014[0][0], p014[0][1],
991       p014[0][2]],
992      [(p014[0][0]+pm[0])/2,
993       (p014[0][1]+pm[1])/2,
994       -90],
995      [pm[0], pm[1], pm[2]]
996  ])
997  for t in np.linspace(
998          tiemposairereposo[0],
999          tiemposairereposo[-1],
1000         num=int((tiemposairereposo[-1] -
1001         tiemposairereposo[0]) * 105)):
1002     pos_t = sc.calcular_qt(
1003         protairep4return,
1004         tiemposairereposo, t)
1005     poses.append((pos_t, t))
1006     theta_t = ink.calcular_angulos(
1007         *pos_t)
1008     angulo_t = np.array([
1009         ((theta_t[0] * 180.0) / np.pi)
1010         + 90,
1011         ((theta_t[1] * 180.0) / np.pi)
1012         + 90,
1013         360 - ((theta_t[2] * 180.0)
1014         / np.pi)
1015     ])
1016     if angulo_t[2] < 0:
1017         angulo_t[2] += 360
1018     if angulo_t[2] > 360:
1019         angulo_t[2] -= 360
1020     us = np.array([
1021         (502.0 / 45.0) * angulo_t[0]
1022         + 464,
1023         (989.0 / 90.0) * angulo_t[1]
1024         + 901.0 / 2.0,
```

```
1025            (98.0 / 9.0) * angulo_t[2]
1026            + 468.0
1027        ])
1028        ticks_t = np.array([
1029            int(us[0] / 4.88),
1030            int(us[1] / 4.88),
1031            int(us[2] / 4.88)
1032        ])
1033        ticks_Rotaire4return.append(
1034            (us.astype(int), t))
1035
1036    # AIRE DE RETURN ROTACION 6
1037    poses = []
1038    ticks_Rotaire6return = []
1039    protairep6return = np.array([
1040        [p036[0][0], p036[0][1],
1041         p036[0][2]],
1042        [(p036[0][0]+pm[0])/2,
1043         (p036[0][1]+pm[1])/2,
1044         -90],
1045        [pm[0], pm[1], pm[2]]
1046    ])
1047    for t in np.linspace(
1048            tiemposairereposo[0],
1049            tiemposairereposo[-1],
1050            num=int((tiemposairereposo[-1] -
1051            tiemposairereposo[0]) * 105)):
1052        pos_t = sc.calcular_qt(
1053            protairep6return,
1054            tiemposairereposo, t)
1055        poses.append((pos_t, t))
1056        theta_t = ink.calcular_angulos(
1057            *pos_t)
1058        angulo_t = np.array([
1059            ((theta_t[0] * 180.0) / np.pi)
1060            + 90,
1061            ((theta_t[1] * 180.0) / np.pi)
1062            + 90,
1063            360 - ((theta_t[2] * 180.0)
1064            / np.pi)
1065        ])
1066        if angulo_t[2] < 0:
1067            angulo_t[2] += 360
1068        if angulo_t[2] > 360:
1069            angulo_t[2] -= 360
1070        us = np.array([
1071            (1021.0 / 90.0) * angulo_t[0]
1072            + 1071.0 / 2.0,
1073            (169.0 / 15.0) * angulo_t[1]
1074            + 490.0,
1075            (496.0 / 45.0) * angulo_t[2]
1076            + 439.0
1077        ])
1078        ticks_t = np.array([
1079            int(us[0] / 4.88),
1080            int(us[1] / 4.88),
1081            int(us[2] / 4.88)
```

```
    ])
    ticks_Rotaire6return.append(
        (us.astype(int), t))

# FUNCION PARA GUARDAR TICKS EN TXT
def escribir_lita_entxt(
        thetas, ruta_archivo):
    with open(ruta_archivo, 'w',
              newline='') as archivo:
        for array, valor_adicional in
thetas:
            linea = list(array)
            +[valor_adicional]
            linea_texto = ' '.join(
                map(str, linea))
            archivo.write(linea_texto +
                          '\n')

# GENERAR FICHERO CON LOS TICKS
thetas = []
for i, theta_t in enumerate(
        ticks_Rotarrastre2):
    t = theta_t[1]
    aux = np.concatenate((
        ticks_Rotarrastre1[i][0],
        ticks_Rotaire2[i][0],
        ticks_Rotarrastre3[i][0],
        ticks_Rotaire4[i][0],
        ticks_Rotarrastre5[i][0],
        ticks_Rotaire6[i][0]))
    thetas.append((aux, t))
for i, theta_t in enumerate(
        ticks_Rotarrastre2):
    t = theta_t[1]
    aux = np.concatenate((
        ticks_Rotaire1[i][0],
        ticks_Rotarrastre2[i][0],
        ticks_Rotaire3[i][0],
        ticks_Rotarrastre4[i][0],
        ticks_Rotaire5[i][0],
        ticks_Rotarrastre6[i][0]))
    thetas.append((aux, t))
escribir_lita_entxt(thetas,
                    'rotation.txt')

# GENERAR FICHERO TICKS DESDE REPOSO
thetas = []
for i, theta_t in enumerate(
        ticks_Rotarrastre2reposo):
    t = theta_t[1]
    aux = np.concatenate((
        ticks_Rotaire1reposo[i][0],
        ticks_Rotarrastre2reposo[i][0],
        ticks_Rotaire3reposo[i][0],
        ticks_Rotarrastre4reposo[i][0],
        ticks_Rotaire5reposo[i][0],
        ticks_Rotarrastre6reposo[i][0]))
```

```
1139      thetas.append((aux, t))
1140  escribir_lita_entxt(thetas,
1141                      'rotreposo.txt')
1142
1143  # GENERAR FICHERO TICKS RETURN
1144  thetas = []
1145  for i, theta_t in enumerate(
1146          ticks_Rotarrastre1return):
1147      t = theta_t[1]
1148      aux = np.concatenate((
1149          ticks_Rotarrastre1return[i][0],
1150          ticks_Rotaire2return[i][0],
1151          ticks_Rotarrastre3return[i][0],
1152          ticks_Rotaire4return[i][0],
1153          ticks_Rotarrastre5return[i][0],
1154          ticks_Rotaire6return[i][0]))
1155      thetas.append((aux, t))
1156  escribir_lita_entxt(thetas,
1157                      'rotreturn.txt')
```

10.8.8 movimiento_lineal.py

```
import trapezoidal_interpolation as trapint
import inverse_kinematics as ink
import process_data as pdata
import spline_cubico as sc
import numpy as np

# AJUSTAR TAMAÑO DEL PASO
semipasoy = 50
centroy = 0
alturaz = -114
separacionx = 126
pm = np.array([separacionx, centroy, alturaz])

# T para el paso
T = 2
Treposo = 1
tiemposaire = np.array([0, T/2, T])
tiemposairereposo = np.array([0, Treposo/2, Treposo])

# PUNTOS PATAS 90°
p090 = np.array([separacionx, pm[1] - semipasoy, alturaz])
pf90 = np.array([separacionx, pm[1] + semipasoy, alturaz])
paire90p2 = np.array([
    [pf90[0], pf90[1], pf90[2]],
    [pm[0], pm[1], -80],
    [p090[0], p090[1], p090[2]]
])
paire90p5 = np.array([
    [p090[0], p090[1], p090[2]],
    [pm[0], pm[1], -80],
    [pf90[0], pf90[1], pf90[2]]
])
paire90p5reposo = np.array([
    [pm[0], pm[1], pm[2]],
    [pm[0], (pf90[1] + pm[1]) / 2, -90],
    [pf90[0], pf90[1], pf90[2]]
])
paire90p2return = np.array([
    [pf90[0], pf90[1], pf90[2]],
    [pm[0], (pf90[1] + pm[1]) / 2, -90],
    [pm[0], pm[1], pm[2]]
])

# PUNTOS PATAS 45°
p0mas45 = np.array([
    separacionx + semipasoy * np.cos(np.pi/4),
    -semipasoy * np.sin(np.pi/4),
    alturaz
])
pfmas45 = np.array([
    separacionx - semipasoy * np.cos(np.pi/4),
    semipasoy * np.sin(np.pi/4),
    alturaz
])
p0menos45 = np.array([
```

```
56      separacionx - semipasoy * np.cos(np.pi/4),
57      -semipasoy * np.sin(np.pi/4),
58      alturaz
59  ])
60  pfmenos45 = np.array([
61      separacionx + semipasoy * np.cos(np.pi/4),
62      semipasoy * np.sin(np.pi/4),
63      alturaz
64  ])
65  paire45p1 = np.array([
66      [pfmas45[0], pfmas45[1], pfmas45[2]],
67      [pm[0], pm[1], -80],
68      [p0mas45[0], p0mas45[1], p0mas45[2]]
69  ])
70  paire45p3 = np.array([
71      [pfmenos45[0], pfmenos45[1], pfmenos45[2]],
72      [pm[0], pm[1], -80],
73      [p0menos45[0], p0menos45[1], p0menos45[2]]
74  ])
75  paire45p4 = np.array([
76      [p0mas45[0], p0mas45[1], p0mas45[2]],
77      [pm[0], pm[1], -80],
78      [pfmas45[0], pfmas45[1], pfmas45[2]]
79  ])
80  paire45p6 = np.array([
81      [p0menos45[0], p0menos45[1], p0menos45[2]],
82      [pm[0], pm[1], -80],
83      [pfmenos45[0], pfmenos45[1], pfmenos45[2]]
84  ])
85  paire45p1reposo = np.array([
86      [pm[0], pm[1], pm[2]],
87      [(p0mas45[0] + pm[0]) / 2, (p0mas45[1] + pm[1]) / 2, -90],
88      [p0mas45[0], p0mas45[1], p0mas45[2]]
89  ])
90  paire45p3reposo = np.array([
91      [pm[0], pm[1], pm[2]],
92      [(p0menos45[0] + pm[0]) / 2, (p0menos45[1] + pm[1]) / 2, -
93  90],
94      [p0menos45[0], p0menos45[1], p0menos45[2]]
95  ])
96  paire45p4return = np.array([
97      [p0mas45[0], p0mas45[1], p0mas45[2]],
98      [(p0mas45[0] + pm[0]) / 2, (p0mas45[1] + pm[1]) / 2, -90],
99      [pm[0], pm[1], pm[2]]
100 ])
101 paire45p6return = np.array([
102     [p0menos45[0], p0menos45[1], p0menos45[2]],
103     [(p0menos45[0] + pm[0]) / 2, (p0menos45[1] + pm[1]) / 2, -
104 90],
105     [pm[0], pm[1], pm[2]]
106 ])
107
108 # PATAS QUE FORMAN 90° (2 Y 5)
109 # Angulos Arrastre (PRIMER TRAMO) PATA 2
110 poses = []
111 ticks_arrastre2 = []
112
```

```
113 for t in np.linspace(0, T, num=int(T * 105)):
114     pos_t = trapint.calcular_qt(p090, pf90, t, T)
115     poses.append((pos_t, t))
116     theta_t = ink.calcular_angulos(*pos_t)
117     angulo_t = np.array([
118         ((theta_t[0] * 180.0) / np.pi) + 90,
119         ((theta_t[1] * 180.0) / np.pi) + 90,
120         360 - ((theta_t[2] * 180.0) / np.pi)
121     ])
122     if angulo_t[2] < 0:
123         angulo_t[2] += 360
124     if angulo_t[2] > 360:
125         angulo_t[2] -= 360
126     us = np.array([
127         (205.0 / 18.0) * angulo_t[0] + 1069.0 / 2.0,
128         (313.0 / 30.0) * angulo_t[1] + 1187.0 / 2.0,
129         (199.0 / 18.0) * angulo_t[2] + 1005.0 / 2.0
130     ])
131     ticks_t = np.array([
132         int(us[0] / 4.88),
133         int(us[1] / 4.88),
134         int(us[2] / 4.88)
135     ])
136     ticks_arrastre2.append((ticks_t, t))
137
138 # Angulos Arrastre (PRIMER TRAMO) PATA 5
139 poses = []
140 ticks_arrastre5 = []
141
142 for t in np.linspace(0, T, num=int(T * 105)):
143     pos_t = trapint.calcular_qt(pf90, p090, t, T)
144     poses.append((pos_t, t))
145     theta_t = ink.calcular_angulos(*pos_t)
146     angulo_t = np.array([
147         ((theta_t[0] * 180.0) / np.pi) + 90,
148         ((theta_t[1] * 180.0) / np.pi) + 90,
149         360 - ((theta_t[2] * 180.0) / np.pi)
150     ])
151     if angulo_t[2] < 0:
152         angulo_t[2] += 360
153     if angulo_t[2] > 360:
154         angulo_t[2] -= 360
155     us = np.array([
156         (361.0 / 30.0) * angulo_t[0] + 661.0 / 2.0,
157         (97.0 / 9.0) * angulo_t[1] + 550.0,
158         (524.0 / 45.0) * angulo_t[2] + 523.0
159     ])
160     ticks_t = np.array([
161         int(us[0] / 4.88),
162         int(us[1] / 4.88),
163         int(us[2] / 4.88)
164     ])
165     ticks_arrastre5.append((ticks_t, t))
166
167 # Angulos Aire (SEGUNDO TRAMO) PATA 2
168 poses = []
169 ticks_aire2 = []
```

```
170
171 for t in np.linspace(tiemposaire[0], tiemposaire[-1],
172                        num=int((tiemposaire[-1] - tiemposaire[0]) *
173 105)):
174     pos_t = sc.calcular_qt(paire90p2, tiemposaire, t)
175     poses.append((pos_t, t))
176     theta_t = ink.calcular_angulos(*pos_t)
177     angulo_t = np.array([
178         ((theta_t[0] * 180.0) / np.pi) + 90,
179         ((theta_t[1] * 180.0) / np.pi) + 90,
180         360 - ((theta_t[2] * 180.0) / np.pi)
181     ])
182     if angulo_t[2] < 0:
183         angulo_t[2] += 360
184     if angulo_t[2] > 360:
185         angulo_t[2] -= 360
186     us = np.array([
187         (205.0 / 18.0) * angulo_t[0] + 1069.0 / 2.0,
188         (313.0 / 30.0) * angulo_t[1] + 1187.0 / 2.0,
189         (199.0 / 18.0) * angulo_t[2] + 1005.0 / 2.0
190     ])
191     ticks_t = np.array([
192         int(us[0] / 4.88),
193         int(us[1] / 4.88),
194         int(us[2] / 4.88)
195     ])
196     ticks_aire2.append((ticks_t, t))
197
198 # Angulos Aire (SEGUNDO TRAMO) PATA 5
199 poses = []
200 ticks_aire5 = []
201
202 for t in np.linspace(tiemposaire[0], tiemposaire[-1],
203                        num=int((tiemposaire[-1] - tiemposaire[0]) *
204 105)):
205     pos_t = sc.calcular_qt(paire90p5, tiemposaire, t)
206     poses.append((pos_t, t))
207     theta_t = ink.calcular_angulos(*pos_t)
208     angulo_t = np.array([
209         ((theta_t[0] * 180.0) / np.pi) + 90,
210         ((theta_t[1] * 180.0) / np.pi) + 90,
211         360 - ((theta_t[2] * 180.0) / np.pi)
212     ])
213     if angulo_t[2] < 0:
214         angulo_t[2] += 360
215     if angulo_t[2] > 360:
216         angulo_t[2] -= 360
217     us = np.array([
218         (361.0 / 30.0) * angulo_t[0] + 661.0 / 2.0,
219         (97.0 / 9.0) * angulo_t[1] + 550.0,
220         (524.0 / 45.0) * angulo_t[2] + 523.0
221     ])
222     ticks_t = np.array([
223         int(us[0] / 4.88),
224         int(us[1] / 4.88),
225         int(us[2] / 4.88)
226     ])
```

```
227     ticks_aire5.append((ticks_t, t))
228 
229 # ANGULOS DESDE REPOSO Y RETURN 90° PATAS 2 Y 5
230 # Angulos Paso desde Reposo-Arrastre Pata 2
231 poses = []
232 ticks_arrastre2reposo = []
233 
234 for t in np.linspace(0, Treposo, num=int(Treposo * 105)):
235     pos_t = trapint.calcular_qt(pm, pf90, t, Treposo)
236     poses.append((pos_t, t))
237     theta_t = ink.calcular_angulos(*pos_t)
238     angulo_t = np.array([
239         ((theta_t[0] * 180.0) / np.pi) + 90,
240         ((theta_t[1] * 180.0) / np.pi) + 90,
241         360 - ((theta_t[2] * 180.0) / np.pi)
242     ])
243     if angulo_t[2] < 0:
244         angulo_t[2] += 360
245     if angulo_t[2] > 360:
246         angulo_t[2] -= 360
247     us = np.array([
248         (205.0 / 18.0) * angulo_t[0] + 1069.0 / 2.0,
249         (313.0 / 30.0) * angulo_t[1] + 1187.0 / 2.0,
250         (199.0 / 18.0) * angulo_t[2] + 1005.0 / 2.0
251     ])
252     ticks_t = np.array([
253         int(us[0] / 4.88),
254         int(us[1] / 4.88),
255         int(us[2] / 4.88)
256     ])
257     ticks_arrastre2reposo.append((ticks_t, t))
258 
259 # Angulos Paso desde Reposo-Aire Pata 5
260 poses = []
261 ticks_aire5reposo = []
262 
263 for t in np.linspace(tiemposaire[0], tiemposairereposo[-1],
264                      num=int((tiemposairereposo[-1] -
265 tiemposairereposo[0]) * 105)):
266     pos_t = sc.calcular_qt(paire90p5reposo, tiemposairereposo, t)
267     poses.append((pos_t, t))
268     theta_t = ink.calcular_angulos(*pos_t)
269     angulo_t = np.array([
270         ((theta_t[0] * 180.0) / np.pi) + 90,
271         ((theta_t[1] * 180.0) / np.pi) + 90,
272         360 - ((theta_t[2] * 180.0) / np.pi)
273     ])
274     if angulo_t[2] < 0:
275         angulo_t[2] += 360
276     if angulo_t[2] > 360:
277         angulo_t[2] -= 360
278     us = np.array([
279         (361.0 / 30.0) * angulo_t[0] + 661.0 / 2.0,
280         (97.0 / 9.0) * angulo_t[1] + 550.0,
281         (524.0 / 45.0) * angulo_t[2] + 523.0
282     ])
283     ticks_t = np.array([
```

```
284         int(us[0] / 4.88),
285         int(us[1] / 4.88),
286         int(us[2] / 4.88)
287     ])
288     ticks_aire5reposo.append((ticks_t, t))
289 
290 # Angulos Paso Return-Aire Pata 2
291 poses = []
292 ticks_aire2return = []
293 
294 for t in np.linspace(tiemposairereposo[0], tiemposairereposo[-1],
295                      num=int((tiemposairereposo[-1] -
296 tiemposairereposo[0]) * 105)):
297     pos_t = sc.calcular_qt(paire90p2return, tiemposairereposo, t)
298     poses.append((pos_t, t))
299     theta_t = ink.calcular_angulos(*pos_t)
300     angulo_t = np.array([
301         ((theta_t[0] * 180.0) / np.pi) + 90,
302         ((theta_t[1] * 180.0) / np.pi) + 90,
303         360 - ((theta_t[2] * 180.0) / np.pi)
304     ])
305     if angulo_t[2] < 0:
306         angulo_t[2] += 360
307     if angulo_t[2] > 360:
308         angulo_t[2] -= 360
309     us = np.array([
310         (205.0 / 18.0) * angulo_t[0] + 1069.0 / 2.0,
311         (313.0 / 30.0) * angulo_t[1] + 1187.0 / 2.0,
312         (199.0 / 18.0) * angulo_t[2] + 1005.0 / 2.0
313     ])
314     ticks_t = np.array([
315         int(us[0] / 4.88),
316         int(us[1] / 4.88),
317         int(us[2] / 4.88)
318     ])
319     ticks_aire2return.append((ticks_t, t))
320 
321 # Angulos Paso Return-Arrastre Pata 5
322 poses = []
323 ticks_arrastre5return = []
324 
325 for t in np.linspace(0, Treposo, num=int(Treposo * 105)):
326     pos_t = trapint.calcular_qt(pf90, pm, t, Treposo)
327     poses.append((pos_t, t))
328     theta_t = ink.calcular_angulos(*pos_t)
329     angulo_t = np.array([
330         ((theta_t[0] * 180.0) / np.pi) + 90,
331         ((theta_t[1] * 180.0) / np.pi) + 90,
332         360 - ((theta_t[2] * 180.0) / np.pi)
333     ])
334     if angulo_t[2] < 0:
335         angulo_t[2] += 360
336     if angulo_t[2] > 360:
337         angulo_t[2] -= 360
338     us = np.array([
339         (361.0 / 30.0) * angulo_t[0] + 661.0 / 2.0,
340         (97.0 / 9.0) * angulo_t[1] + 550.0,
```

```
341             (524.0 / 45.0) * angulo_t[2] + 523.0
342         ])
343         ticks_t = np.array([
344             int(us[0] / 4.88),
345             int(us[1] / 4.88),
346             int(us[2] / 4.88)
347         ])
348         ticks_arrastre5return.append((ticks_t, t))
349 
350 # PATAS QUE FORMAN 45° (1, 3, 4 Y 6)
351 # Angulos Arrastre (PRIMER TRAMO) PATA 1
352 poses1 = []
353 ticks_arrastre1 = []
354 
355 for t in np.linspace(0, T, num=int(T * 105)):
356     pos_t = trapint.calcular_qt(p0mas45, pfmas45, t, T)
357     poses1.append((pos_t, t))
358     theta_t = ink.calcular_angulos(*pos_t)
359     angulo_t = np.array([
360         ((theta_t[0] * 180.0) / np.pi) + 90,
361         ((theta_t[1] * 180.0) / np.pi) + 90,
362         360 - ((theta_t[2] * 180.0) / np.pi)
363     ])
364     if angulo_t[2] < 0:
365         angulo_t[2] += 360
366     if angulo_t[2] > 360:
367         angulo_t[2] -= 360
368     us = np.array([
369         (983.0 / 90.0) * angulo_t[0] + 863.0 / 2.0,
370         (989.0 / 90.0) * angulo_t[1] + 937.0 / 2.0,
371         (1079.0 / 90.0) * angulo_t[2] + 683.0 / 2.0
372     ])
373     ticks_t = np.array([
374         int(us[0] / 4.88),
375         int(us[1] / 4.88),
376         int(us[2] / 4.88)
377     ])
378     ticks_arrastre1.append((ticks_t, t))
379 
380 pdata.grafica_p(poses1)
381 
382 #--------------Angulos Arrastre (PRIMER TRAMO) PATA 3------------
383 ---
384 poses = []
385 ticks_arrastre3 = []
386 
387 for t in np.linspace(0, T, num=int(T*105)):
388     pos_t = trapint.calcular_qt(p0menos45, pfmenos45, t, T)
389     poses.append((pos_t, t))
390     theta_t = ink.calcular_angulos(*pos_t)
391     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
392                          ((theta_t[1] * 180.0) / np.pi) + 90,
393                          360 - ((theta_t[2] * 180.0) / np.pi)])
394     if angulo_t[2] < 0:
395         angulo_t[2] += 360
396     if angulo_t[2] > 360:
397         angulo_t[2] -= 360
```

```
398     us = np.array([(496.0 / 45.0) * angulo_t[0] + 514.0,
399                    (959.0 / 90.0) * angulo_t[1] + 1029.0 / 2.0,
400                    (491.0 / 45.0) * angulo_t[2] + 507])
401     ticks_t = np.array([int(us[0] / 4.88),
402                         int(us[1] / 4.88),
403                         int(us[2] / 4.88)])
404     ticks_arrastre3.append((ticks_t, t))
405 pdata.grafica_p(poses)
406 
407 #------- Angulos Arrastre (PRIMER TRAMO) PATA 4----------
408 poses = []
409 ticks_arrastre4 = []
410 
411 for t in np.linspace(0, T, num=int(T*105)):
412     pos_t = trapint.calcular_qt(pfmas45, p0mas45, t, T)
413     poses.append((pos_t, t))
414     theta_t = ink.calcular_angulos(*pos_t)
415     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
416                          ((theta_t[1] * 180.0) / np.pi) + 90,
417                          360 - ((theta_t[2] * 180.0) / np.pi)])
418     if angulo_t[2] < 0:
419         angulo_t[2] += 360
420     if angulo_t[2] > 360:
421         angulo_t[2] -= 360
422     us = np.array([(502.0 / 45.0) * angulo_t[0] + 464,
423                    (989.0 / 90.0) * angulo_t[1] + 901.0 / 2.0,
424                    (98.0 / 9.0) * angulo_t[2] + 468.0])
425     ticks_t = np.array([int(us[0] / 4.88),
426                         int(us[1] / 4.88),
427                         int(us[2] / 4.88)])
428     ticks_arrastre4.append((ticks_t, t))
429 pdata.grafica_p(poses)
430 
431 #---------- Angulos Arrastre (PRIMER TRAMO) PATA 6--------------
432 poses = []
433 ticks_arrastre6 = []
434 
435 for t in np.linspace(0, T, num=int(T*105)):
436     pos_t = trapint.calcular_qt(pfmenos45, p0menos45, t, T)
437     poses.append((pos_t, t))
438     theta_t = ink.calcular_angulos(*pos_t)
439     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
440                          ((theta_t[1] * 180.0) / np.pi) + 90,
441                          360 - ((theta_t[2] * 180.0) / np.pi)])
442     if angulo_t[2] < 0:
443         angulo_t[2] += 360
444     if angulo_t[2] > 360:
445         angulo_t[2] -= 360
446     us = np.array([(1021.0 / 90.0) * angulo_t[0] + 1071.0 / 2.0,
447                    (169.0 / 15.0) * angulo_t[1] + 490.0,
448                    (496.0 / 45.0) * angulo_t[2] + 439.0])
449     ticks_t = np.array([int(us[0] / 4.88),
450                         int(us[1] / 4.88),
451                         int(us[2] / 4.88)])
452     ticks_arrastre6.append((ticks_t, t))
453 pdata.grafica_p(poses)
454 
```

```
455 #------------ Angulos Aire (SEGUNDO TRAMO) PATA 1------------
456 poses = []
457 ticks_aire1 = []
458 
459 for t in np.linspace(tiemposaire[0], tiemposaire[-1],
460                      num=int((tiemposaire[-1] - tiemposaire[0]) *
461 105)):
462     pos_t = sc.calcular_qt(paire45p1, tiemposaire, t)
463     poses.append((pos_t, t))
464     theta_t = ink.calcular_angulos(*pos_t)
465     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
466                          ((theta_t[1] * 180.0) / np.pi) + 90,
467                          360 - ((theta_t[2] * 180.0) / np.pi)])
468     if angulo_t[2] < 0:
469         angulo_t[2] += 360
470     if angulo_t[2] > 360:
471         angulo_t[2] -= 360
472     us = np.array([(983.0 / 90.0) * angulo_t[0] + 863.0 / 2.0,
473                    (989.0 / 90.0) * angulo_t[1] + 937.0 / 2.0,
474                    (1079.0 / 90.0) * angulo_t[2] + 683.0 / 2.0])
475     ticks_t = np.array([int(us[0] / 4.88),
476                         int(us[1] / 4.88),
477                         int(us[2] / 4.88)])
478     ticks_aire1.append((ticks_t, t))
479 
480 #-------------- Angulos Aire (SEGUNDO TRAMO) PATA 3--------------
481 -
482 poses = []
483 ticks_aire3 = []
484 
485 for t in np.linspace(tiemposaire[0], tiemposaire[-1],
486                      num=int((tiemposaire[-1] - tiemposaire[0]) *
487 105)):
488     pos_t = sc.calcular_qt(paire45p3, tiemposaire, t)
489     poses.append((pos_t, t))
490     theta_t = ink.calcular_angulos(*pos_t)
491     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
492                          ((theta_t[1] * 180.0) / np.pi) + 90,
493                          360 - ((theta_t[2] * 180.0) / np.pi)])
494     if angulo_t[2] < 0:
495         angulo_t[2] += 360
496     if angulo_t[2] > 360:
497         angulo_t[2] -= 360
498     us = np.array([(496.0 / 45.0) * angulo_t[0] + 514.0,
499                    (959.0 / 90.0) * angulo_t[1] + 1029.0 / 2.0,
500                    (491.0 / 45.0) * angulo_t[2] + 507])
501     ticks_t = np.array([int(us[0] / 4.88),
502                         int(us[1] / 4.88),
503                         int(us[2] / 4.88)])
504     ticks_aire3.append((ticks_t, t))
505 
506 #------------ Angulos Aire (SEGUNDO TRAMO) PATA 4--------------
507 ----
508 poses = []
509 ticks_aire4 = []
510 
511 for t in np.linspace(tiemposaire[0], tiemposaire[-1],
```

```
512                     num=int((tiemposaire[-1] - tiemposaire[0]) *
513 105)):
514     pos_t = sc.calcular_qt(paire45p4, tiemposaire, t)
515     poses.append((pos_t, t))
516     theta_t = ink.calcular_angulos(*pos_t)
517     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
518                          ((theta_t[1] * 180.0) / np.pi) + 90,
519                          360 - ((theta_t[2] * 180.0) / np.pi)])
520     if angulo_t[2] < 0:
521         angulo_t[2] += 360
522     if angulo_t[2] > 360:
523         angulo_t[2] -= 360
524     us = np.array([(502.0 / 45.0) * angulo_t[0] + 464,
525                    (989.0 / 90.0) * angulo_t[1] + 901.0 / 2.0,
526                    (98.0 / 9.0) * angulo_t[2] + 468.0])
527     ticks_t = np.array([int(us[0] / 4.88),
528                         int(us[1] / 4.88),
529                         int(us[2] / 4.88)])
530     ticks_aire4.append((ticks_t, t))
531
532 #-------------- Angulos Aire (SEGUNDO TRAMO) PATA 6--------------
533 -
534 poses = []
535 ticks_aire6 = []
536
537 for t in np.linspace(tiemposaire[0], tiemposaire[-1],
538                      num=int((tiemposaire[-1] - tiemposaire[0]) *
539 105)):
540     pos_t = sc.calcular_qt(paire45p6, tiemposaire, t)
541     poses.append((pos_t, t))
542     theta_t = ink.calcular_angulos(*pos_t)
543     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
544                          ((theta_t[1] * 180.0) / np.pi) + 90,
545                          360 - ((theta_t[2] * 180.0) / np.pi)])
546     if angulo_t[2] < 0:
547         angulo_t[2] += 360
548     if angulo_t[2] > 360:
549         angulo_t[2] -= 360
550     us = np.array([(1021.0 / 90.0) * angulo_t[0] + 1071.0 / 2.0,
551                    (169.0 / 15.0) * angulo_t[1] + 490.0,
552                    (496.0 / 45.0) * angulo_t[2] + 439.0])
553     ticks_t = np.array([int(us[0] / 4.88),
554                         int(us[1] / 4.88),
555                         int(us[2] / 4.88)])
556     ticks_aire6.append((ticks_t, t))
557
558 '''''
559 ---------ANGULOS DESDE REPOSO Y RETURN 45° PATAS 1,3,4 Y 6-------
560 -----
561 '''''
562
563 #------------ ANGULOS PASO DESDE REPOSO-ARRASTRE PATA 4----------
564 ---
565 poses = []
566 ticks_arrastre4reposo = []
567
568 for t in np.linspace(0, Treposo, num=int(Treposo * 105)):
```

```
569     pos_t = trapint.calcular_qt(pm, p0mas45, t, Treposo)
570     poses.append((pos_t, t))
571     theta_t = ink.calcular_angulos(*pos_t)
572     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
573                          ((theta_t[1] * 180.0) / np.pi) + 90,
574                          360 - ((theta_t[2] * 180.0) / np.pi)])
575     if angulo_t[2] < 0:
576         angulo_t[2] += 360
577     if angulo_t[2] > 360:
578         angulo_t[2] -= 360
579     us = np.array([(502.0 / 45.0) * angulo_t[0] + 464,
580                    (989.0 / 90.0) * angulo_t[1] + 901.0 / 2.0,
581                    (98.0 / 9.0) * angulo_t[2] + 468.0])
582     ticks_t = np.array([int(us[0] / 4.88),
583                         int(us[1] / 4.88),
584                         int(us[2] / 4.88)])
585     ticks_arrastre4reposo.append((ticks_t, t))
586 
587 #---------- ANGULOS PASO DESDE REPOSO-ARRASTRE PATA 6---------
588 poses = []
589 ticks_arrastre6reposo = []
590 
591 for t in np.linspace(0, Treposo, num=int(Treposo * 105)):
592     pos_t = trapint.calcular_qt(pm, p0menos45, t, Treposo)
593     poses.append((pos_t, t))
594     theta_t = ink.calcular_angulos(*pos_t)
595     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
596                          ((theta_t[1] * 180.0) / np.pi) + 90,
597                          360 - ((theta_t[2] * 180.0) / np.pi)])
598     if angulo_t[2] < 0:
599         angulo_t[2] += 360
600     if angulo_t[2] > 360:
601         angulo_t[2] -= 360
602     us = np.array([(1021.0 / 90.0) * angulo_t[0] + 1071.0 / 2.0,
603                    (169.0 / 15.0) * angulo_t[1] + 490.0,
604                    (496.0 / 45.0) * angulo_t[2] + 439.0])
605     ticks_t = np.array([int(us[0] / 4.88),
606                         int(us[1] / 4.88),
607                         int(us[2] / 4.88)])
608     ticks_arrastre6reposo.append((ticks_t, t))
609 
610 #--------- ANGULOS PASO DESDE REPOSO-AIRE PATA 1-----------
611 poses = []
612 ticks_aire1reposo = []
613 
614 for t in np.linspace(tiemposairereposo[0],
615                      tiemposairereposo[-1],
616                      num=int((tiemposairereposo[-1] -
617                               tiemposairereposo[0]) * 105)):
618     pos_t = sc.calcular_qt(paire45p1reposo,
619                            tiemposairereposo, t)
620     poses.append((pos_t, t))
621     theta_t = ink.calcular_angulos(*pos_t)
622     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
623                          ((theta_t[1] * 180.0) / np.pi) + 90,
624                          360 - ((theta_t[2] * 180.0) / np.pi)])
625     if angulo_t[2] < 0:
```

```
626         angulo_t[2] += 360
627     if angulo_t[2] > 360:
628         angulo_t[2] -= 360
629     us = np.array([(983.0 / 90.0) * angulo_t[0] + 863.0 / 2.0,
630                    (989.0 / 90.0) * angulo_t[1] + 937.0 / 2.0,
631                    (1079.0 / 90.0) * angulo_t[2] + 683.0 / 2.0])
632     ticks_t = np.array([int(us[0] / 4.88),
633                         int(us[1] / 4.88),
634                         int(us[2] / 4.88)])
635     ticks_aire1reposo.append((ticks_t, t))
636 
637 #----------- ANGULOS PASO DESDE REPOSO-AIRE PATA 3-----------
638 poses = []
639 ticks_aire3reposo = []
640 
641 for t in np.linspace(tiemposairereposo[0],
642                      tiemposairereposo[-1],
643                      num=int((tiemposairereposo[-1] -
644                               tiemposairereposo[0]) * 105)):
645     pos_t = sc.calcular_qt(paire45p3reposo,
646                            tiemposairereposo, t)
647     poses.append((pos_t, t))
648     theta_t = ink.calcular_angulos(*pos_t)
649     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
650                          ((theta_t[1] * 180.0) / np.pi) + 90,
651                          360 - ((theta_t[2] * 180.0) / np.pi)])
652     if angulo_t[2] < 0:
653         angulo_t[2] += 360
654     if angulo_t[2] > 360:
655         angulo_t[2] -= 360
656     us = np.array([(496.0 / 45.0) * angulo_t[0] + 514.0,
657                    (959.0 / 90.0) * angulo_t[1] + 1029.0 / 2.0,
658                    (491.0 / 45.0) * angulo_t[2] + 507])
659     ticks_t = np.array([int(us[0] / 4.88),
660                         int(us[1] / 4.88),
661                         int(us[2] / 4.88)])
662     ticks_aire3reposo.append((ticks_t, t))
663 
664 #--------- ANGULOS PASO RETURN-AIRE PATA 4---------
665 poses = []
666 ticks_aire4return = []
667 
668 for t in np.linspace(tiemposairereposo[0],
669                      tiemposairereposo[-1],
670                      num=int((tiemposairereposo[-1] -
671                               tiemposairereposo[0]) * 105)):
672     pos_t = sc.calcular_qt(paire45p4return,
673                            tiemposairereposo, t)
674     poses.append((pos_t, t))
675     theta_t = ink.calcular_angulos(*pos_t)
676     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
677                          ((theta_t[1] * 180.0) / np.pi) + 90,
678                          360 - ((theta_t[2] * 180.0) / np.pi)])
679     if angulo_t[2] < 0:
680         angulo_t[2] += 360
681     if angulo_t[2] > 360:
682         angulo_t[2] -= 360
```

```
683     us = np.array([(502.0 / 45.0) * angulo_t[0] + 464,
684                    (989.0 / 90.0) * angulo_t[1] + 901.0 / 2.0,
685                    (98.0 / 9.0) * angulo_t[2] + 468.0])
686     ticks_t = np.array([int(us[0] / 4.88),
687                         int(us[1] / 4.88),
688                         int(us[2] / 4.88)])
689     ticks_aire4return.append((ticks_t, t))
690 
691 #--------- ANGULOS PASO RETURN-AIRE PATA 6----------
692 poses = []
693 ticks_aire6return = []
694 
695 for t in np.linspace(tiemposairereposo[0],
696                      tiemposairereposo[-1],
697                      num=int((tiemposairereposo[-1] -
698                               tiemposairereposo[0]) * 105)):
699     pos_t = sc.calcular_qt(paire45p6return,
700                            tiemposairereposo, t)
701     poses.append((pos_t, t))
702     theta_t = ink.calcular_angulos(*pos_t)
703     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
704                          ((theta_t[1] * 180.0) / np.pi) + 90,
705                          360 - ((theta_t[2] * 180.0) / np.pi)])
706     if angulo_t[2] < 0:
707         angulo_t[2] += 360
708     if angulo_t[2] > 360:
709         angulo_t[2] -= 360
710     us = np.array([(1021.0 / 90.0) * angulo_t[0] + 1071.0 / 2.0,
711                    (169.0 / 15.0) * angulo_t[1] + 490.0,
712                    (496.0 / 45.0) * angulo_t[2] + 439.0])
713     ticks_t = np.array([int(us[0] / 4.88),
714                         int(us[1] / 4.88),
715                         int(us[2] / 4.88)])
716     ticks_aire6return.append((ticks_t, t))
717 
718 #---------- ANGULOS PASO RETURN-ARRASTRE PATA 1---------
719 poses1 = []
720 ticks_arrastre1return = []
721 
722 for t in np.linspace(0, Treposo, num=int(Treposo * 105)):
723     pos_t = trapint.calcular_qt(p0mas45, pm, t, Treposo)
724     poses1.append((pos_t, t))
725     theta_t = ink.calcular_angulos(*pos_t)
726     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
727                          ((theta_t[1] * 180.0) / np.pi) + 90,
728                          360 - ((theta_t[2] * 180.0) / np.pi)])
729     if angulo_t[2] < 0:
730         angulo_t[2] += 360
731     if angulo_t[2] > 360:
732         angulo_t[2] -= 360
733     us = np.array([(983.0 / 90.0) * angulo_t[0] + 863.0 / 2.0,
734                    (989.0 / 90.0) * angulo_t[1] + 937.0 / 2.0,
735                    (1079.0 / 90.0) * angulo_t[2] + 683.0 / 2.0])
736     ticks_t = np.array([int(us[0] / 4.88),
737                         int(us[1] / 4.88),
738                         int(us[2] / 4.88)])
739     ticks_arrastre1return.append((ticks_t, t))
```

```
740
741 #---------- ANGULOS PASO RETURN-ARRASTRE PATA 3------------
742 poses = []
743 ticks_arrastre3return = []
744
745 for t in np.linspace(0, Treposo, num=int(Treposo * 105)):
746     pos_t = trapint.calcular_qt(p0menos45, pm, t, Treposo)
747     poses.append((pos_t, t))
748     theta_t = ink.calcular_angulos(*pos_t)
749     angulo_t = np.array([((theta_t[0] * 180.0) / np.pi) + 90,
750                          ((theta_t[1] * 180.0) / np.pi) + 90,
751                          360 - ((theta_t[2] * 180.0) / np.pi)])
752     if angulo_t[2] < 0:
753         angulo_t[2] += 360
754     if angulo_t[2] > 360:
755         angulo_t[2] -= 360
756     us = np.array([(496.0 / 45.0) * angulo_t[0] + 514.0,
757                    (959.0 / 90.0) * angulo_t[1] + 1029.0 / 2.0,
758                    (491.0 / 45.0) * angulo_t[2] + 507])
759     ticks_t = np.array([int(us[0] / 4.88),
760                         int(us[1] / 4.88),
761                         int(us[2] / 4.88)])
762     ticks_arrastre3return.append((ticks_t, t))
763
764 #--------FUNCION PARA GUARDAR TICKS EN TXT -------------
765 def escribir_lita_entxt(thetas, ruta_archivo):
766     with open(ruta_archivo, 'w', newline='') as archivo:
767         for array, valor_adicional in thetas:
768             linea = list(array) + [valor_adicional]
769             linea_texto = ' '.join(map(str, linea))
770             archivo.write(linea_texto + '\n')
771
772
773 '''''
774 ------------GENERAR FICHERO CON LOS TICKS PARA EL PASO F Y B-----
775 --------
776 '''''
777 thetas = []
778 for i, theta_t in enumerate(ticks_arrastre2):
779     t = theta_t[1]
780     aux = np.concatenate((ticks_arrastre1[i][0],
781 ticks_aire2[i][0],
782                           ticks_arrastre3[i][0],
783 ticks_aire4[i][0],
784                           ticks_arrastre5[i][0],
785 ticks_aire6[i][0]))
786     thetas.append((aux, t))
787
788 #----------SEGUNDO TRAMO----------------
789 for i, theta_t in enumerate(ticks_arrastre2):
790     t = theta_t[1]
791     aux = np.concatenate((ticks_aire1[i][0],
792 ticks_arrastre2[i][0],
793                           ticks_aire3[i][0],
794 ticks_arrastre4[i][0],
795                           ticks_aire5[i][0],
796 ticks_arrastre6[i][0]))
```

```
797     thetas.append((aux, t))
798 escribir_lita_entxt(thetas, 'forward.txt')
799 
800 '''''
801 ----------GENERAR FICHERO CON LOS TICKS PARA EL PASO DESDE
802 REPOSO---------------
803 '''''
804 thetas = []
805 for i, theta_t in enumerate(ticks_arrastre2reposo):
806     t = theta_t[1]
807     aux = np.concatenate((ticks_aire1reposo[i][0],
808 ticks_arrastre2reposo[i][0],
809                           ticks_aire3reposo[i][0],
810 ticks_arrastre4reposo[i][0],
811                           ticks_aire5reposo[i][0],
812 ticks_arrastre6reposo[i][0]))
813     thetas.append((aux, t))
814 escribir_lita_entxt(thetas, 'reposo.txt')
815 
816 '''''
817 ------------GENERAR FICHERO CON LOS TICKS PARA EL PASO DE RETURN-
818 ---------------
819 '''''
820 thetas = []
821 for i, theta_t in enumerate(ticks_arrastre1return):
822     t = theta_t[1]
823     aux = np.concatenate((ticks_arrastre1return[i][0],
824 ticks_aire2return[i][0],
825                           ticks_arrastre3return[i][0],
826 ticks_aire4return[i][0],
827                           ticks_arrastre5return[i][0],
828 ticks_aire6return[i][0]))
829     thetas.append((aux, t))
830 escribir_lita_entxt(thetas, 'return.txt')
```

10.8.9 Pasomodificado.py

```
import trapezoidal_interpolation as trapint
import inverse_kinematics as ink
import process_data as pdata
import spline_cubico as sc
import numpy as np
import pickle
import os

# CREAR CARPETA PASO
carpeta = 'Thetas_paso'
if not os.path.exists(carpeta):
    os.makedirs(carpeta)

# AJUSTAR TAMAÑO PASO
# Definir punto inicio/final
semipasoy = 50 # mm semipaso
centroy = 0 # mm centro
alturaz = -114 # mm suelo
separacionx = 126 # mm
airealtua = -65 # mm
pm = np.array([separacionx, centroy,
               alturaz])

# PUNTOS PATAS 90°
p090 = np.array([separacionx,
                 pm[1] - semipasoy,
                 alturaz])
pf90 = np.array([separacionx,
                 pm[1] + semipasoy,
                 alturaz])

paire90p2 = np.array([
    [pf90[0], pf90[1], pf90[2]],
    [pm[0], pm[1], airealtua],
    [p090[0], p090[1], p090[2]]
])

paire90p5 = np.array([
    [p090[0], p090[1], p090[2]],
    [pm[0], pm[1], airealtua],
    [pf90[0], pf90[1], pf90[2]]
])

paire90p5reposo = np.array([
    [pm[0], pm[1], pm[2]],
    [pm[0], (pf90[1] + pm[1]) / 2,
     airealtua],
    [pf90[0], pf90[1], pf90[2]]
])

paire90p2return = np.array([
    [pf90[0], pf90[1], pf90[2]],
    [pm[0], (pf90[1] + pm[1]) / 2,
     airealtua],
    [pm[0], pm[1], pm[2]]
```

```
56 ])
57
58 # PUNTOS PATAS 45°
59 p0mas45 = np.array([separacionx +
60                     semipasoy * np.cos(np.pi / 4),
61                     -semipasoy * np.sin(np.pi / 4),
62                     alturaz])
63 pfmas45 = np.array([separacionx -
64                     semipasoy * np.cos(np.pi / 4),
65                     semipasoy * np.sin(np.pi / 4),
66                     alturaz])
67
68 p0menos45 = np.array([separacionx -
69                       semipasoy * np.cos(np.pi /
70 4),
71                       -semipasoy * np.sin(np.pi /
72 4),
73                       alturaz])
74 pfmenos45 = np.array([separacionx +
75                       semipasoy * np.cos(np.pi /
76 4),
77                       semipasoy * np.sin(np.pi /
78 4),
79                       alturaz])
80
81 paire45p1 = np.array([
82     [pfmas45[0], pfmas45[1], pfmas45[2]],
83     [pm[0], pm[1], airealtua],
84     [p0mas45[0], p0mas45[1], p0mas45[2]]
85 ])
86
87 paire45p3 = np.array([
88     [pfmenos45[0], pfmenos45[1],
89      pfmenos45[2]],
90     [pm[0], pm[1], airealtua],
91     [p0menos45[0], p0menos45[1],
92      p0menos45[2]]
93 ])
94
95 paire45p4 = np.array([
96     [p0mas45[0], p0mas45[1],
97      p0mas45[2]],
98     [pm[0], pm[1], airealtua],
99     [pfmas45[0], pfmas45[1],
100     pfmas45[2]]
101 ])
102
103 paire45p6 = np.array([
104     [p0menos45[0], p0menos45[1],
105      p0menos45[2]],
106     [pm[0], pm[1], airealtua],
107     [pfmenos45[0], pfmenos45[1],
108      pfmenos45[2]]
109 ])
110
111 paire45p1reposo = np.array([
112     [pm[0], pm[1], pm[2]],
```

```
113      [(p0mas45[0] + pm[0]) / 2,
114       (p0mas45[1] + pm[1]) / 2,
115       airealtua],
116      [p0mas45[0], p0mas45[1],
117       p0mas45[2]]
118  ])
119
120  paire45p3reposo = np.array([
121      [pm[0], pm[1], pm[2]],
122      [(p0menos45[0] + pm[0]) / 2,
123       (p0menos45[1] + pm[1]) / 2,
124       airealtua],
125      [p0menos45[0], p0menos45[1],
126       p0menos45[2]]
127  ])
128
129  paire45p4return = np.array([
130      [p0mas45[0], p0mas45[1],
131       p0mas45[2]],
132      [(p0mas45[0] + pm[0]) / 2,
133       (p0mas45[1] + pm[1]) / 2,
134       airealtua],
135      [pm[0], pm[1], pm[2]]
136  ])
137
138  paire45p6return = np.array([
139      [p0menos45[0], p0menos45[1],
140       p0menos45[2]],
141      [(p0menos45[0] + pm[0]) / 2,
142       (p0menos45[1] + pm[1]) / 2,
143       airealtua],
144      [pm[0], pm[1], pm[2]]
145  ])
146
147  # PATAS 90° (2 Y 5)
148  T = 2 # s tiempo final
149  Treposo = 1 # s tiempo final reposo
150  tiemposaire = np.array([0, T / 2, T])
151  tiemposairereposo = np.array([0,
152                                Treposo / 2,
153                                Treposo])
154
155  # Angulos Arrastre (PRIMER TRAMO)
156  # PATA 2
157  poses = []
158  ticks_arrastre2 = []
159
160  for t in np.linspace(0, T, num=int(T *
161                                     105)):
162      pos_t = trapint.calcular_qt(p090,
163                                  pf90,
164                                  t, T)
165      poses.append((pos_t, t))
166      theta_t = ink.calcular_angulos(*pos_t)
167      angulo_t = np.array([((theta_t[0] *
168                             180.0) / np.pi) + 90,
169                           ((theta_t[1] * 180.0)
```

```
                          / np.pi) + 90,
                         360 - ((theta_t[2] *
                                 180.0) / np.pi)])
    if angulo_t[2] < 0:
        angulo_t[2] += 360
    if angulo_t[2] > 360:
        angulo_t[2] -= 360
    us = np.array([(205.0 / 18.0) *
                   angulo_t[0] + 1069.0
                   / 2.0,
                   (313.0 / 30.0) *
                   angulo_t[1] + 1187.0
                   / 2.0,
                   (199.0 / 18.0) *
                   angulo_t[2] + 1005.0
                   / 2.0])
    ticks_t = np.array([int(us[0] / 4.88),
                        int(us[1] / 4.88),
                        int(us[2] / 4.88)])
    ticks_arrastre2.append((ticks_t,
                            t))

# Angulos Arrastre (PRIMER TRAMO)
# PATA 5
poses = []
ticks_arrastre5 = []

for t in np.linspace(0, T, num=int(T *
                                   105)):
    pos_t = trapint.calcular_qt(pf90,
                                p090,
                                t, T)
    poses.append((pos_t, t))
    theta_t = ink.calcular_angulos(*pos_t)
    angulo_t = np.array([((theta_t[0] *
                           180.0) / np.pi) + 90,
                         ((theta_t[1] * 180.0)
                          / np.pi) + 90,
                         360 - ((theta_t[2] *
                                 180.0) / np.pi)])
    if angulo_t[2] < 0:
        angulo_t[2] += 360
    if angulo_t[2] > 360:
        angulo_t[2] -= 360
    us = np.array([(361.0 / 30.0) *
                   angulo_t[0] + 661.0
                   / 2.0,
                   (97.0 / 9.0) *
                   angulo_t[1] + 550.0,
                   (524.0 / 45.0) *
                   angulo_t[2] + 523.0])
    ticks_t = np.array([int(us[0] / 4.88),
                        int(us[1] / 4.88),
                        int(us[2] / 4.88)])
    ticks_arrastre5.append((ticks_t,
                            t))

```

```
227 # Angulos Aire (SEGUNDO TRAMO)
228 # PATA 2
229 poses = []
230 ticks_aire2 = []
231
232 for t in np.linspace(tiemposaire[0],
233                      tiemposaire[-1],
234                      num=int((tiemposaire[-1] -
235                               tiemposaire[0]) *
236                              105)):
237     pos_t = sc.calcular_qt(paire90p2,
238                            tiemposaire, t)
239     poses.append((pos_t, t))
240     theta_t = ink.calcular_angulos(*pos_t)
241     angulo_t = np.array([((theta_t[0] *
242                            180.0) / np.pi) + 90,
243                          ((theta_t[1] * 180.0)
244                           / np.pi) + 90,
245                          360 - ((theta_t[2] *
246                                  180.0) / np.pi)])
247     if angulo_t[2] < 0:
248         angulo_t[2] += 360
249     if angulo_t[2] > 360:
250         angulo_t[2] -= 360
251     us = np.array([(205.0 / 18.0) *
252                    angulo_t[0] + 1069.0
253                    / 2.0,
254                    (313.0 / 30.0) *
255                    angulo_t[1] + 1187.0
256                    / 2.0,
257                    (199.0 / 18.0) *
258                    angulo_t[2] + 1005.0
259                    / 2.0])
260     ticks_t = np.array([int(us[0] / 4.88),
261                         int(us[1] / 4.88),
262                         int(us[2] / 4.88)])
263     ticks_aire2.append((ticks_t, t))
264
265 # Angulos Aire (SEGUNDO TRAMO)
266 # PATA 5
267 poses = []
268 ticks_aire5 = []
269
270 for t in np.linspace(tiemposaire[0],
271                      tiemposaire[-1],
272                      num=int((tiemposaire[-1] -
273                               tiemposaire[0]) *
274                              105)):
275     pos_t = sc.calcular_qt(paire90p5,
276                            tiemposaire, t)
277     poses.append((pos_t, t))
278     theta_t = ink.calcular_angulos(*pos_t)
279     angulo_t = np.array([((theta_t[0] *
280                            180.0) / np.pi) + 90,
281                          ((theta_t[1] * 180.0)
282                           / np.pi) + 90,
283                          360 - ((theta_t[2] *
```

```
                                    180.0) / np.pi)])
    if angulo_t[2] < 0:
        angulo_t[2] += 360
    if angulo_t[2] > 360:
        angulo_t[2] -= 360
    us = np.array([(361.0 / 30.0) *
                   angulo_t[0] + 661.0
                   / 2.0,
                   (97.0 / 9.0) *
                   angulo_t[1] + 550.0,
                   (524.0 / 45.0) *
                   angulo_t[2] + 523.0])
    ticks_t = np.array([int(us[0] / 4.88),
                        int(us[1] / 4.88),
                        int(us[2] / 4.88)])
    ticks_aire5.append((ticks_t, t))

# ANGULOS DESDE REPOSO Y RETURN
# 90° PATAS 2 Y 5
# ANGULOS PASO DESDE REPOSO-ARRASTRE
# PATA 2
poses = []
ticks_arrastre2reposo = []

for t in np.linspace(0, Treposo,
                     num=int(Treposo *
                             105)):
    pos_t = trapint.calcular_qt(pm,
                                pf90, t,
                                Treposo)
    poses.append((pos_t, t))
    theta_t = ink.calcular_angulos(*pos_t)
    angulo_t = np.array([((theta_t[0] *
                           180.0) / np.pi) + 90,
                         ((theta_t[1] * 180.0)
                          / np.pi) + 90,
                         360 - ((theta_t[2] *
                                 180.0) / np.pi)])
    if angulo_t[2] < 0:
        angulo_t[2] += 360
    if angulo_t[2] > 360:
        angulo_t[2] -= 360
    us = np.array([(205.0 / 18.0) *
                   angulo_t[0] + 1069.0
                   / 2.0,
                   (313.0 / 30.0) *
                   angulo_t[1] + 1187.0
                   / 2.0,
                   (199.0 / 18.0) *
                   angulo_t[2] + 1005.0
                   / 2.0])
    ticks_t = np.array([int(us[0] / 4.88),
                        int(us[1] / 4.88),
                        int(us[2] / 4.88)])
    ticks_arrastre2reposo.append((ticks_t,
                                  t))

```

```
341 # ANGULOS PASO DESDE REPOSO-AIRE
342 # PATA 5
343 poses = []
344 ticks_aire5reposo = []
345
346 for t in np.linspace(tiemposaire[0],
347                      tiemposairereposo[-1],
348                      num=int((tiemposairereposo[-1]
349 -
350                               tiemposairereposo[0])
351 *
352                              105)):
353     pos_t = sc.calcular_qt(paire90p5reposo,
354                            tiemposairereposo,
355                            t)
356     poses.append((pos_t, t))
357     theta_t = ink.calcular_angulos(*pos_t)
358     angulo_t = np.array([((theta_t[0] *
359                            180.0) / np.pi) + 90,
360                          ((theta_t[1] * 180.0)
361                           / np.pi) + 90,
362                          360 - ((theta_t[2] *
363                                  180.0) / np.pi)])
364     if angulo_t[2] < 0:
365         angulo_t[2] += 360
366     if angulo_t[2] > 360:
367         angulo_t[2] -= 360
368     us = np.array([(361.0 / 30.0) *
369                    angulo_t[0] + 661.0
370                    / 2.0,
371                    (97.0 / 9.0) *
372                    angulo_t[1] + 550.0,
373                    (524.0 / 45.0) *
374                    angulo_t[2] + 523.0])
375     ticks_t = np.array([int(us[0] / 4.88),
376                         int(us[1] / 4.88),
377                         int(us[2] / 4.88)])
378     ticks_aire5reposo.append((ticks_t,
379                               t))
380
381 # ANGULOS PASO RETURN-AIRE
382 # PATA 2
383 poses = []
384 ticks_aire2return = []
385
386 for t in np.linspace(tiemposairereposo[0],
387                      tiemposairereposo[-1],
388                      num=int((tiemposairereposo[-1]
389 -
390                               tiemposairereposo[0])
391 *
392                              105)):
393     pos_t = sc.calcular_qt(paire90p2return,
394                            tiemposairereposo,
395                            t)
396     poses.append((pos_t, t))
397     theta_t = ink.calcular_angulos(*pos_t)
```

```
398     angulo_t = np.array([((theta_t[0] *
399                            180.0) / np.pi) + 90,
400                           ((theta_t[1] * 180.0)
401                            / np.pi) + 90,
402                           360 - ((theta_t[2] *
403                                   180.0) / np.pi)])
404     if angulo_t[2] < 0:
405         angulo_t[2] += 360
406     if angulo_t[2] > 360:
407         angulo_t[2] -= 360
408     us = np.array([(205.0 / 18.0) *
409                    angulo_t[0] + 1069.0
410                    / 2.0,
411                    (313.0 / 30.0) *
412                    angulo_t[1] + 1187.0
413                    / 2.0,
414                    (199.0 / 18.0) *
415                    angulo_t[2] + 1005.0
416                    / 2.0])
417     ticks_t = np.array([int(us[0] / 4.88),
418                         int(us[1] / 4.88),
419                         int(us[2] / 4.88)])
420     ticks_aire2return.append((ticks_t,
421                               t))
422
423 # ANGULOS PASO RETURN-ARRASTRE
424 # PATA 5
425 poses = []
426 ticks_arrastre5return = []
427
428 for t in np.linspace(0, Treposo,
429                      num=int(Treposo *
430                              105)):
431     pos_t = trapint.calcular_qt(pf90,
432                                 pm, t,
433                                 Treposo)
434     poses.append((pos_t, t))
435     theta_t = ink.calcular_angulos(*pos_t)
436     angulo_t = np.array([((theta_t[0] *
437                            180.0) / np.pi) + 90,
438                           ((theta_t[1] * 180.0)
439                            / np.pi) + 90,
440                           360 - ((theta_t[2] *
441                                   180.0) / np.pi)])
442     if angulo_t[2] < 0:
443         angulo_t[2] += 360
444     if angulo_t[2] > 360:
445         angulo_t[2] -= 360
446     us = np.array([(361.0 / 30.0) *
447                    angulo_t[0] + 661.0
448                    / 2.0,
449                    (97.0 / 9.0) *
450                    angulo_t[1] + 550.0,
451                    (524.0 / 45.0) *
452                    angulo_t[2] + 523.0])
453     ticks_t = np.array([int(us[0] / 4.88),
454                         int(us[1] / 4.88),
```

```
455                         int(us[2] / 4.88)])
456     ticks_arrastre5return.append((ticks_t,
457                                   t))
458
459 # PATAS 45° (1, 3, 4 Y 6)
460 # Angulos Arrastre (PRIMER TRAMO)
461 # PATA 1
462 poses1 = []
463 ticks_arrastre1 = []
464
465 for t in np.linspace(0, T, num=int(T *
466                                    105)):
467     pos_t = trapint.calcular_qt(p0mas45,
468                                 pfmas45,
469                                 t, T)
470     poses1.append((pos_t, t))
471     theta_t = ink.calcular_angulos(*pos_t)
472     angulo_t = np.array([((theta_t[0] *
473                            180.0) / np.pi) + 90,
474                           ((theta_t[1] * 180.0)
475                            / np.pi) + 90,
476                           360 - ((theta_t[2] *
477                                   180.0) / np.pi)])
478     if angulo_t[2] < 0:
479         angulo_t[2] += 360
480     if angulo_t[2] > 360:
481         angulo_t[2] -= 360
482     us = np.array([(983.0 / 90.0) *
483                    angulo_t[0] + 863.0
484                    / 2.0,
485                    (989.0 / 90.0) *
486                    angulo_t[1] + 937.0
487                    / 2.0,
488                    (1079.0 / 90.0) *
489                    angulo_t[2] + 683.0
490                    / 2.0])
491     ticks_t = np.array([int(us[0] / 4.88),
492                         int(us[1] / 4.88),
493                         int(us[2] / 4.88)])
494     ticks_arrastre1.append((ticks_t, t))
495
496 pdata.grafica_p(poses1)
497
498 # Angulos Arrastre (PRIMER TRAMO)
499 # PATA 3
500 poses = []
501 ticks_arrastre3 = []
502
503 for t in np.linspace(0, T, num=int(T *
504                                    105)):
505     pos_t = trapint.calcular_qt(p0menos45,
506                                 pfmenos45,
507                                 t, T)
508     poses.append((pos_t, t))
509     theta_t = ink.calcular_angulos(*pos_t)
510     angulo_t = np.array([((theta_t[0] *
511                            180.0) / np.pi) + 90,
```

```
512                          ((theta_t[1] * 180.0)
513                           / np.pi) + 90,
514                          360 - ((theta_t[2] *
515                                  180.0) / np.pi)])
516     if angulo_t[2] < 0:
517         angulo_t[2] += 360
518     if angulo_t[2] > 360:
519         angulo_t[2] -= 360
520     us = np.array([(496.0 / 45.0) *
521                    angulo_t[0] + 514.0,
522                    (959.0 / 90.0) *
523                    angulo_t[1] + 1029.0
524                    / 2.0,
525                    (491.0 / 45.0) *
526                    angulo_t[2] + 507])
527     ticks_t = np.array([int(us[0] / 4.88),
528                         int(us[1] / 4.88),
529                         int(us[2] / 4.88)])
530     ticks_arrastre3.append((ticks_t,
531                             t))
532
533 pdata.grafica_p(poses)
534
535 # Angulos Arrastre (PRIMER TRAMO)
536 # PATA 4
537 poses = []
538 ticks_arrastre4 = []
539
540 for t in np.linspace(0, T, num=int(T *
541                                    105)):
542     pos_t = trapint.calcular_qt(pfmas45,
543                                 p0mas45,
544                                 t, T)
545     poses.append((pos_t, t))
546     theta_t = ink.calcular_angulos(*pos_t)
547     angulo_t = np.array([((theta_t[0] *
548                            180.0) / np.pi) + 90,
549                          ((theta_t[1] * 180.0)
550                           / np.pi) + 90,
551                          360 - ((theta_t[2] *
552                                  180.0) / np.pi)])
553     if angulo_t[2] < 0:
554         angulo_t[2] += 360
555     if angulo_t[2] > 360:
556         angulo_t[2] -= 360
557     us = np.array([(502.0 / 45.0) *
558                    angulo_t[0] + 464,
559                    (989.0 / 90.0) *
560                    angulo_t[1] + 901.0
561                    / 2.0,
562                    (98.0 / 9.0) *
563                    angulo_t[2] + 468.0])
564     ticks_t = np.array([int(us[0] / 4.88),
565                         int(us[1] / 4.88),
566                         int(us[2] / 4.88)])
567     ticks_arrastre4.append((ticks_t,
568                             t))
```

```
569
570 pdata.grafica_p(poses)
571
572 # Angulos Arrastre (PRIMER
573 # TRAMO) PATA 6
574 poses = []
575 ticks_arrastre6 = []
576
577 for t in np.linspace(0, T,
578                      num=int(T * 105)):
579     pos_t = trapint.calcular_qt(
580         pfmenos45, p0menos45, t, T)
581     poses.append((pos_t, t))
582     theta_t = ink.calcular_angulos(
583         *pos_t)
584     angulo_t = np.array([
585         ((theta_t[0] * 180.0) / np.pi)
586         + 90,
587         ((theta_t[1] * 180.0) / np.pi)
588         + 90,
589         360 - ((theta_t[2] * 180.0)
590                / np.pi)])
591     if angulo_t[2] < 0:
592         angulo_t[2] += 360
593     if angulo_t[2] > 360:
594         angulo_t[2] -= 360
595     us = np.array([
596         (1021.0 / 90.0) * angulo_t[0]
597         + 1071.0 / 2.0,
598         (169.0 / 15.0) * angulo_t[1]
599         + 490.0,
600         (496.0 / 45.0) * angulo_t[2]
601         + 439.0])
602     ticks_t = np.array([
603         int(us[0] / 4.88),
604         int(us[1] / 4.88),
605         int(us[2] / 4.88)])
606     ticks_arrastre6.append((ticks_t,
607                             t))
608 pdata.grafica_p(poses)
609
610 # Angulos Aire (SEGUNDO TRAMO)
611 # PATA 1
612 poses = []
613 ticks_aire1 = []
614
615 for t in np.linspace(tiemposaire[0],
616                      tiemposaire[-1],
617                      num=int((tiemposaire[-1]
618                               - tiemposaire[0])
619                              * 105)):
620     pos_t = sc.calcular_qt(paire45p1,
621                            tiemposaire, t)
622     poses.append((pos_t, t))
623     theta_t = ink.calcular_angulos(
624         *pos_t)
625     angulo_t = np.array([
```

```
626             ((theta_t[0] * 180.0) / np.pi)
627             + 90,
628             ((theta_t[1] * 180.0) / np.pi)
629             + 90,
630             360 - ((theta_t[2] * 180.0)
631                    / np.pi)])
632         if angulo_t[2] < 0:
633             angulo_t[2] += 360
634         if angulo_t[2] > 360:
635             angulo_t[2] -= 360
636         us = np.array([
637             (983.0 / 90.0) * angulo_t[0]
638             + 863.0 / 2.0,
639             (989.0 / 90.0) * angulo_t[1]
640             + 937.0 / 2.0,
641             (1079.0 / 90.0) * angulo_t[2]
642             + 683.0 / 2.0])
643         ticks_t = np.array([
644             int(us[0] / 4.88),
645             int(us[1] / 4.88),
646             int(us[2] / 4.88)])
647         ticks_aire1.append((ticks_t,
648                             t))
649
650     # Angulos Aire (SEGUNDO TRAMO)
651     # PATA 3
652     poses = []
653     ticks_aire3 = []
654
655     for t in np.linspace(tiemposaire[0],
656                          tiemposaire[-1],
657                          num=int((tiemposaire[-1]
658                                   - tiemposaire[0])
659                                  * 105)):
660         pos_t = sc.calcular_qt(paire45p3,
661                                tiemposaire, t)
662         poses.append((pos_t, t))
663         theta_t = ink.calcular_angulos(
664             *pos_t)
665         angulo_t = np.array([
666             ((theta_t[0] * 180.0) / np.pi)
667             + 90,
668             ((theta_t[1] * 180.0) / np.pi)
669             + 90,
670             360 - ((theta_t[2] * 180.0)
671                    / np.pi)])
672         if angulo_t[2] < 0:
673             angulo_t[2] += 360
674         if angulo_t[2] > 360:
675             angulo_t[2] -= 360
676         us = np.array([
677             (496.0 / 45.0) * angulo_t[0]
678             + 514.0,
679             (959.0 / 90.0) * angulo_t[1]
680             + 1029.0 / 2.0,
681             (491.0 / 45.0) * angulo_t[2]
682             + 507])
```

```
683     ticks_t = np.array([
684         int(us[0] / 4.88),
685         int(us[1] / 4.88),
686         int(us[2] / 4.88)])
687     ticks_aire3.append((ticks_t,
688                         t))
689
690 # Angulos Aire (SEGUNDO TRAMO)
691 # PATA 4
692 poses = []
693 ticks_aire4 = []
694
695 for t in np.linspace(tiemposaire[0],
696                      tiemposaire[-1],
697                      num=int((tiemposaire[-1]
698                               - tiemposaire[0])
699                              * 105)):
700     pos_t = sc.calcular_qt(paire45p4,
701                            tiemposaire, t)
702     poses.append((pos_t, t))
703     theta_t = ink.calcular_angulos(
704         *pos_t)
705     angulo_t = np.array([
706         ((theta_t[0] * 180.0) / np.pi)
707         + 90,
708         ((theta_t[1] * 180.0) / np.pi)
709         + 90,
710         360 - ((theta_t[2] * 180.0)
711                / np.pi)])
712     if angulo_t[2] < 0:
713         angulo_t[2] += 360
714     if angulo_t[2] > 360:
715         angulo_t[2] -= 360
716     us = np.array([
717         (502.0 / 45.0) * angulo_t[0]
718         + 464,
719         (989.0 / 90.0) * angulo_t[1]
720         + 901.0 / 2.0,
721         (98.0 / 9.0) * angulo_t[2]
722         + 468.0])
723     ticks_t = np.array([
724         int(us[0] / 4.88),
725         int(us[1] / 4.88),
726         int(us[2] / 4.88)])
727     ticks_aire4.append((ticks_t,
728                         t))
729
730 # Angulos Aire (SEGUNDO TRAMO)
731 # PATA 6
732 poses = []
733 ticks_aire6 = []
734
735 for t in np.linspace(tiemposaire[0],
736                      tiemposaire[-1],
737                      num=int((tiemposaire[-1]
738                               - tiemposaire[0])
739                              * 105)):
```

```
740     pos_t = sc.calcular_qt(paire45p6,
741                            tiemposaire, t)
742     poses.append((pos_t, t))
743     theta_t = ink.calcular_angulos(
744         *pos_t)
745     angulo_t = np.array([
746         ((theta_t[0] * 180.0) / np.pi)
747         + 90,
748         ((theta_t[1] * 180.0) / np.pi)
749         + 90,
750         360 - ((theta_t[2] * 180.0)
751                / np.pi)])
752     if angulo_t[2] < 0:
753         angulo_t[2] += 360
754     if angulo_t[2] > 360:
755         angulo_t[2] -= 360
756     us = np.array([
757         (1021.0 / 90.0) * angulo_t[0]
758         + 1071.0 / 2.0,
759         (169.0 / 15.0) * angulo_t[1]
760         + 490.0,
761         (496.0 / 45.0) * angulo_t[2]
762         + 439.0])
763     ticks_t = np.array([
764         int(us[0] / 4.88),
765         int(us[1] / 4.88),
766         int(us[2] / 4.88)])
767     ticks_aire6.append((ticks_t,
768                         t))
769
770 # ANGULOS DESDE REPOSO Y RETURN
771 # 45° PATAS 1,3,4 Y 6
772
773 # ANGULOS PASO DESDE REPOSO-ARRASTRE
774 # PATA 4
775 poses = []
776 ticks_arrastre4reposo = []
777
778 for t in np.linspace(0, Treposo,
779                      num=int(Treposo * 105)):
780     pos_t = trapint.calcular_qt(pm,
781                                 p0mas45, t,
782                                 Treposo)
783     poses.append((pos_t, t))
784     theta_t = ink.calcular_angulos(
785         *pos_t)
786     angulo_t = np.array([
787         ((theta_t[0] * 180.0) / np.pi)
788         + 90,
789         ((theta_t[1] * 180.0) / np.pi)
790         + 90,
791         360 - ((theta_t[2] * 180.0)
792                / np.pi)])
793     if angulo_t[2] < 0:
794         angulo_t[2] += 360
795     if angulo_t[2] > 360:
796         angulo_t[2] -= 360
```

```
797     us = np.array([
798         (502.0 / 45.0) * angulo_t[0]
799         + 464,
800         (989.0 / 90.0) * angulo_t[1]
801         + 901.0 / 2.0,
802         (98.0 / 9.0) * angulo_t[2]
803         + 468.0])
804     ticks_t = np.array([
805         int(us[0] / 4.88),
806         int(us[1] / 4.88),
807         int(us[2] / 4.88)])
808     ticks_arrastre4reposo.append(
809         (ticks_t, t))
810
811 # ANGULOS PASO DESDE REPOSO-ARRASTRE
812 # PATA 6
813 poses = []
814 ticks_arrastre6reposo = []
815
816 for t in np.linspace(0, Treposo,
817                      num=int(Treposo * 105)):
818     pos_t = trapint.calcular_qt(pm,
819                                 p0menos45,
820                                 t,
821                                 Treposo)
822     poses.append((pos_t, t))
823     theta_t = ink.calcular_angulos(
824         *pos_t)
825     angulo_t = np.array([
826         ((theta_t[0] * 180.0) / np.pi)
827         + 90,
828         ((theta_t[1] * 180.0) / np.pi)
829         + 90,
830         360 - ((theta_t[2] * 180.0)
831                / np.pi)])
832     if angulo_t[2] < 0:
833         angulo_t[2] += 360
834     if angulo_t[2] > 360:
835         angulo_t[2] -= 360
836     us = np.array([
837         (1021.0 / 90.0) * angulo_t[0]
838         + 1071.0 / 2.0,
839         (169.0 / 15.0) * angulo_t[1]
840         + 490.0,
841         (496.0 / 45.0) * angulo_t[2]
842         + 439.0])
843     ticks_t = np.array([
844         int(us[0] / 4.88),
845         int(us[1] / 4.88),
846         int(us[2] / 4.88)])
847     ticks_arrastre6reposo.append(
848         (ticks_t, t))
849
850 # ANGULOS PASO DESDE REPOSO-AIRE
851 # PATA 1
852 poses = []
853 ticks_aire1reposo = []
```

```
854
855 for t in np.linspace(
856         tiemposairereposo[0],
857         tiemposairereposo[-1],
858         num=int((tiemposairereposo[-1]
859                  - tiemposairereposo[0])
860                 * 105)):
861     pos_t = sc.calcular_qt(
862         paire45p1reposo,
863         tiemposairereposo, t)
864     poses.append((pos_t, t))
865     theta_t = ink.calcular_angulos(
866         *pos_t)
867     angulo_t = np.array([
868         ((theta_t[0] * 180.0) / np.pi)
869         + 90,
870         ((theta_t[1] * 180.0) / np.pi)
871         + 90,
872         360 - ((theta_t[2] * 180.0)
873                / np.pi)])
874     if angulo_t[2] < 0:
875         angulo_t[2] += 360
876     if angulo_t[2] > 360:
877         angulo_t[2] -= 360
878     us = np.array([
879         (983.0 / 90.0) * angulo_t[0]
880         + 863.0 / 2.0,
881         (989.0 / 90.0) * angulo_t[1]
882         + 937.0 / 2.0,
883         (1079.0 / 90.0) * angulo_t[2]
884         + 683.0 / 2.0])
885     ticks_t = np.array([
886         int(us[0] / 4.88),
887         int(us[1] / 4.88),
888         int(us[2] / 4.88)])
889     ticks_aire1reposo.append((ticks_t,
890                               t))
891
892 # ANGULOS PASO DESDE REPOSO-AIRE
893 # PATA 3
894 poses = []
895 ticks_aire3reposo = []
896
897 for t in np.linspace(
898         tiemposairereposo[0],
899         tiemposairereposo[-1],
900         num=int((tiemposairereposo[-1]
901                  - tiemposairereposo[0])
902                 * 105)):
903     pos_t = sc.calcular_qt(
904         paire45p3reposo,
905         tiemposairereposo, t)
906     poses.append((pos_t, t))
907     theta_t = ink.calcular_angulos(
908         *pos_t)
909     angulo_t = np.array([
910         ((theta_t[0] * 180.0) / np.pi)
```

```
911         + 90,
912         ((theta_t[1] * 180.0) / np.pi)
913         + 90,
914         360 - ((theta_t[2] * 180.0)
915                / np.pi)])
916     if angulo_t[2] < 0:
917         angulo_t[2] += 360
918     if angulo_t[2] > 360:
919         angulo_t[2] -= 360
920     us = np.array([
921         (496.0 / 45.0) * angulo_t[0]
922         + 514.0,
923         (959.0 / 90.0) * angulo_t[1]
924         + 1029.0 / 2.0,
925         (491.0 / 45.0) * angulo_t[2]
926         + 507])
927     ticks_t = np.array([
928         int(us[0] / 4.88),
929         int(us[1] / 4.88),
930         int(us[2] / 4.88)])
931     ticks_aire3reposo.append(
932         (ticks_t, t))
933
934 # ANGULOS PASO RETURN-AIRE
935 # PATA 4
936 poses = []
937 ticks_aire4return = []
938
939 for t in np.linspace(
940         tiemposairereposo[0],
941         tiemposairereposo[-1],
942         num=int((tiemposairereposo[-1]
943                  - tiemposairereposo[0])
944                 * 105)):
945     pos_t = sc.calcular_qt(
946         paire45p4return,
947         tiemposairereposo, t)
948     poses.append((pos_t, t))
949     theta_t = ink.calcular_angulos(
950         *pos_t)
951     angulo_t = np.array([
952         ((theta_t[0] * 180.0) / np.pi)
953         + 90,
954         ((theta_t[1] * 180.0) / np.pi)
955         + 90,
956         360 - ((theta_t[2] * 180.0)
957                / np.pi)])
958     if angulo_t[2] < 0:
959         angulo_t[2] += 360
960     if angulo_t[2] > 360:
961         angulo_t[2] -= 360
962     us = np.array([
963         (502.0 / 45.0) * angulo_t[0]
964         + 464,
965         (989.0 / 90.0) * angulo_t[1]
966         + 901.0 / 2.0,
967         (98.0 / 9.0) * angulo_t[2]
```

```
968             + 468.0])
969         ticks_t = np.array([
970             int(us[0] / 4.88),
971             int(us[1] / 4.88),
972             int(us[2] / 4.88)])
973         ticks_aire4return.append(
974             (ticks_t, t))
975
976     # ANGULOS PASO RETURN-AIRE
977     # PATA 6
978     poses = []
979     ticks_aire6return = []
980
981     for t in np.linspace(
982             tiemposairereposo[0],
983             tiemposairereposo[-1],
984             num=int((tiemposairereposo[-1]
985                      - tiemposairereposo[0])
986                     * 105)):
987         pos_t = sc.calcular_qt(
988             paire45p6return,
989             tiemposairereposo, t)
990         poses.append((pos_t, t))
991         theta_t = ink.calcular_angulos(
992             *pos_t)
993         angulo_t = np.array([
994             ((theta_t[0] * 180.0) / np.pi)
995             + 90,
996             ((theta_t[1] * 180.0) / np.pi)
997             + 90,
998             360 - ((theta_t[2] * 180.0)
999                    / np.pi)])
1000        if angulo_t[2] < 0:
1001            angulo_t[2] += 360
1002        if angulo_t[2] > 360:
1003            angulo_t[2] -= 360
1004        us = np.array([
1005            (1021.0 / 90.0) * angulo_t[0]
1006            + 1071.0 / 2.0,
1007            (169.0 / 15.0) * angulo_t[1]
1008            + 490.0,
1009            (496.0 / 45.0) * angulo_t[2]
1010            + 439.0])
1011        ticks_t = np.array([
1012            int(us[0] / 4.88),
1013            int(us[1] / 4.88),
1014            int(us[2] / 4.88)])
1015        ticks_aire6return.append(
1016            (ticks_t, t))
1017
1018    # ANGULOS PASO RETURN-ARRASTRE
1019    # PATA 1
1020    poses1 = []
1021    ticks_arrastre1return = []
1022
1023    for t in np.linspace(0, Treposo,
1024                         num=int(Treposo * 105)):
```

```
1025      pos_t = trapint.calcular_qt(
1026          p0mas45, pm, t, Treposo)
1027      poses1.append((pos_t, t))
1028      theta_t = ink.calcular_angulos(
1029          *pos_t)
1030      angulo_t = np.array([
1031          ((theta_t[0] * 180.0) / np.pi)
1032          + 90,
1033          ((theta_t[1] * 180.0) / np.pi)
1034          + 90,
1035          360 - ((theta_t[2] * 180.0)
1036                 / np.pi)])
1037      if angulo_t[2] < 0:
1038          angulo_t[2] += 360
1039      if angulo_t[2] > 360:
1040          angulo_t[2] -= 360
1041      us = np.array([
1042          (983.0 / 90.0) * angulo_t[0]
1043          + 863.0 / 2.0,
1044          (989.0 / 90.0) * angulo_t[1]
1045          + 937.0 / 2.0,
1046          (1079.0 / 90.0) * angulo_t[2]
1047          + 683.0 / 2.0])
1048      ticks_t = np.array([
1049          int(us[0] / 4.88),
1050          int(us[1] / 4.88),
1051          int(us[2] / 4.88)])
1052      ticks_arrastre1return.append(
1053          (ticks_t, t))
1054
1055  # ANGULOS PASO RETURN-ARRASTRE
1056  # PATA 3
1057  poses = []
1058  ticks_arrastre3return = []
1059
1060  for t in np.linspace(0, Treposo,
1061                       num=int(Treposo * 105)):
1062      pos_t = trapint.calcular_qt(
1063          p0menos45, pm, t, Treposo)
1064      poses.append((pos_t, t))
1065      theta_t = ink.calcular_angulos(
1066          *pos_t)
1067      angulo_t = np.array([
1068          ((theta_t[0] * 180.0) / np.pi)
1069          + 90,
1070          ((theta_t[1] * 180.0) / np.pi)
1071          + 90,
1072          360 - ((theta_t[2] * 180.0)
1073                 / np.pi)])
1074      if angulo_t[2] < 0:
1075          angulo_t[2] += 360
1076      if angulo_t[2] > 360:
1077          angulo_t[2] -= 360
1078      us = np.array([
1079          (496.0 / 45.0) * angulo_t[0]
1080          + 514.0,
1081          (959.0 / 90.0) * angulo_t[1]
```

```
1082          + 1029.0 / 2.0,
1083          (491.0 / 45.0) * angulo_t[2]
1084          + 507])
1085      ticks_t = np.array([
1086          int(us[0] / 4.88),
1087          int(us[1] / 4.88),
1088          int(us[2] / 4.88)])
1089      ticks_arrastre3return.append(
1090          (ticks_t, t))
1091
1092  # FUNCION PARA GUARDAR TICKS EN
1093  # TXT
1094  def escribir_lita_entxt(thetas,
1095                          ruta_archivo):
1096      with open(ruta_archivo, 'w',
1097                newline='') as archivo:
1098          for array, valor_adicional in thetas:
1099              linea = list(array)
1100              + [valor_adicional]
1101              linea_texto = ' '.join(
1102                  map(str, linea))
1103              archivo.write(linea_texto
1104                            + '\n')
1105
1106  # GENERAR FICHERO CON TICKS PARA
1107  # EL PASO F Y B
1108  thetas = []
1109  for i, theta_t in enumerate(
1110          ticks_arrastre2):
1111      t = theta_t[1]
1112      aux = np.concatenate(
1113          (ticks_arrastre1[i][0],
1114           ticks_aire2[i][0],
1115           ticks_arrastre3[i][0],
1116           ticks_aire4[i][0],
1117           ticks_arrastre5[i][0],
1118           ticks_aire6[i][0]))
1119      thetas.append((aux, t))
1120
1121  # SEGUNDO TRAMO
1122  for i, theta_t in enumerate(
1123          ticks_arrastre2):
1124      t = theta_t[1]
1125      aux = np.concatenate(
1126          (ticks_aire1[i][0],
1127           ticks_arrastre2[i][0],
1128           ticks_aire3[i][0],
1129           ticks_arrastre4[i][0],
1130           ticks_aire5[i][0],
1131           ticks_aire6[i][0]))
1132      thetas.append((aux, t))
1133  escribir_lita_entxt(thetas,
1134                      'forwardobs.txt')
1135
1136  # GENERAR FICHERO CON TICKS PARA
1137  # EL PASO DESDE REPOSO
1138  thetas = []
```

```
1139 for i, theta_t in enumerate(
1140         ticks_arrastre2reposo):
1141     t = theta_t[1]
1142     aux = np.concatenate(
1143         (ticks_aire1reposo[i][0],
1144          ticks_arrastre2reposo[i][0],
1145          ticks_aire3reposo[i][0],
1146          ticks_arrastre4reposo[i][0],
1147          ticks_aire5reposo[i][0],
1148          ticks_arrastre6reposo[i][0]))
1149     thetas.append((aux, t))
1150 escribir_lita_entxt(thetas,
1151                     'reposoupobs.txt')
1152
1153 # GENERAR FICHERO CON TICKS PARA
1154 # EL PASO DE RETURN
1155 thetas = []
1156 for i, theta_t in enumerate(
1157         ticks_arrastre1return):
1158     t = theta_t[1]
1159     aux = np.concatenate(
1160         (ticks_arrastre1return[i][0],
1161          ticks_aire2return[i][0],
1162          ticks_arrastre3return[i][0],
1163          ticks_aire4return[i][0],
1164          ticks_arrastre5return[i][0],
1165          ticks_aire6return[i][0]))
1166     thetas.append((aux, t))
1167 escribir_lita_entxt(thetas,
1168                     'returnupobs.txt')
```

10.8.10 hexa.ino

```
#include "SPIFFS.h"
#include <Adafruit_PWMServoDriver.h>
#include "BluetoothSerial.h"

BluetoothSerial SerialBT;
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver(0x41);
Adafruit_PWMServoDriver pwm1 = Adafruit_PWMServoDriver(0x40);

// ------------Declaración de variables--------------
// Variables para la lectura del BT
char dato;
char previo = 0;
// Variables para el cambio de modo
// de paso
char fileName1[] = "/forwardobs.txt";
char fileName2[] = "/reposoobs.txt";
char fileName3[] = "/returnobs.txt";

// Matrices que contendrán las trayectorias
const int rowsA = 105;
const int colsA = 18;
uint16_t matrixRP[rowsA][colsA];
uint16_t matrixRN[rowsA][colsA];
uint16_t matrixROTRP[rowsA][colsA];
uint16_t matrixROTRN[rowsA][colsA];

const int rows = 420;
const int cols = 18;
uint16_t matrixFW[rows][cols];
uint16_t matrixRT[rows][cols];

// Variables para enviar los ticks
// a los módulos
int P1ticks1, P1ticks2, P1ticks3;
int P2ticks1, P2ticks2, P2ticks3;
int P3ticks1, P3ticks2, P3ticks3;
int P4ticks1, P4ticks2, P4ticks3;
int P5ticks1, P5ticks2, P5ticks3;
int P6ticks1, P6ticks2, P6ticks3;

void setup() {
  Wire.begin();
  Wire.setClock(1000000);
  Serial.begin(115200);
  SerialBT.begin("HEXAPOD_ESP32");  // BT device name
  Serial.println("BT started");

  if (!SPIFFS.begin(true)) {
    Serial.println("Error al montar SPIFFS");
    return;
  }

  // Abrir el archivo ROTACION
```

```
54    File fileR = SPIFFS.open("/rotation.txt", "r");
55    if (!fileR) {
56      Serial.println("Error al abrir el archivo");
57      return;
58    }
59    // Leer el archivo ROTACION
60    int row = 0;
61    while (fileR.available() && row < rows) {
62      String line = fileR.readStringUntil('\n');
63      parseLine(line, row, &matrixRT[0][0], cols);
64      row++;
65    }
66    fileR.close();
67
68    // Abrir el archivo REPOSO DE ROTACION
69    fileR = SPIFFS.open("/rotreposo.txt", "r");
70    if (!fileR) {
71      Serial.println("Error al abrir el archivo");
72      return;
73    }
74    // Leer el archivo REPOSO DE ROTACION
75    row = 0;
76    while (fileR.available() && row < rowsA) {
77      String line = fileR.readStringUntil('\n');
78      parseLine(line, row, &matrixROTRP[0][0], cols);
79      row++;
80    }
81    fileR.close();
82
83    // Abrir el archivo ROTACION RETURN
84    fileR = SPIFFS.open("/rotreturn.txt", "r");
85    if (!fileR) {
86      Serial.println("Error al abrir el archivo");
87      return;
88    }
89    // Leer el archivo ROTACION RETURN
90    row = 0;
91    while (fileR.available() && row < rowsA) {
92      String line = fileR.readStringUntil('\n');
93      parseLine(line, row, &matrixROTRN[0][0], cols);
94      row++;
95    }
96    fileR.close();
97
98    cambiomodo('N');
99    pwm.begin();
100   pwm.setPWMFreq(50);
101   delay(10);
102   pwm1.begin();
103   pwm1.setPWMFreq(50);
104 }
105
106 // Función para introducir los datos
107 // leídos a las matrices de trayectorias
108 void parseLine(String line, int row,
109                uint16_t* matrix, int cols) {
110   int col = 0;
```

```
111     int start = 0;
112     for (int i = 0; i < line.length(); i++) {
113       if (line[i] == ' ') {
114         String valueStr = line.substring(start, i);
115         int valueint = valueStr.toInt();
116         matrix[row * cols + col] = (uint16_t)valueint;
117         col++;
118         start = i + 1;
119       }
120     }
121 
122     // Agregar el último valor de la línea
123     if (start < line.length()) {
124       String valueStr = line.substring(start);
125       int valueint = valueStr.toInt();
126       matrix[row * cols + col] = (uint16_t)valueint;
127     }
128   }
129 
130   // Función para modificar el modo de caminar del hexapod
131   void cambiomodo(char modo) {
132     if (modo == 'N') {
133       strcpy(fileName1, "/forward.txt");
134       strcpy(fileName2, "/reposo.txt");
135       strcpy(fileName3, "/return.txt");
136     }
137     if (modo == 'U') {
138       strcpy(fileName1, "/forwardup.txt");
139       strcpy(fileName2, "/reposoup.txt");
140       strcpy(fileName3, "/returnup.txt");
141     }
142     if (modo == 'O') {
143       strcpy(fileName1, "/forwardobs.txt");
144       strcpy(fileName2, "/reposoobs.txt");
145       strcpy(fileName3, "/returnobs.txt");
146     }
147 
148     // Abrir el archivo 1
149     File fileF = SPIFFS.open(fileName1, "r");
150     if (!fileF) {
151       Serial.println("Error al abrir el archivo 1");
152       return;
153     }
154     // Leer el archivo 1
155     int row = 0;
156     while (fileF.available() && row < rows) {
157       String line = fileF.readStringUntil('\n');
158       parseLine(line, row, &matrixFW[0][0], cols);
159       row++;
160     }
161     fileF.close();
162 
163     // Abrir el archivo 2
164     File fileR = SPIFFS.open(fileName2, "r");
165     if (!fileR) {
166       Serial.println("Error al abrir el archivo 2");
167       return;
```

```
168     }
169     // Leer el archivo 2
170     row = 0;
171     while (fileR.available() && row < rowsA) {
172       String line = fileR.readStringUntil('\n');
173       parseLine(line, row, &matrixRP[0][0], cols);
174       row++;
175     }
176     fileR.close();
177 
178     // Abrir el archivo 3
179     Serial.println(fileName3);
180     File fileN = SPIFFS.open(fileName3, "r");
181     if (!fileN) {
182       Serial.println("Error al abrir el archivo 3");
183       return;
184     }
185     // Leer el archivo 3
186     row = 0;
187     while (fileN.available() && row < rowsA) {
188       String line = fileN.readStringUntil('\n');
189       parseLine(line, row, &matrixRN[0][0], cols);
190       row++;
191     }
192     fileN.close();
193   }
194 
195   // Función para enviar los ticks a los servomotores
196   void enviaticks(int P1ticks1, int P1ticks2, int P1ticks3,
197                   int P2ticks1, int P2ticks2, int P2ticks3,
198                   int P3ticks1, int P3ticks2, int P3ticks3,
199                   int P4ticks1, int P4ticks2, int P4ticks3,
200                   int P5ticks1, int P5ticks2, int P5ticks3,
201                   int P6ticks1, int P6ticks2, int P6ticks3) {
202     // PATA 1
203     pwm.setPWM(1, 0, P1ticks1 + 17);
204     pwm.setPWM(2, 0, P1ticks2 + 17);
205     pwm.setPWM(3, 0, P1ticks3 + 17);
206     // PATA 2
207     pwm.setPWM(4, 0, P2ticks1 + 17);
208     pwm.setPWM(5, 0, P2ticks2 + 17);
209     pwm.setPWM(6, 0, P2ticks3 + 17);
210     // PATA 3
211     pwm.setPWM(7, 0, P3ticks1 + 17);
212     pwm.setPWM(8, 0, P3ticks2 + 17);
213     pwm.setPWM(9, 0, P3ticks3 + 17);
214     // PATA 4
215     pwm1.setPWM(1, 0, P4ticks1);
216     pwm1.setPWM(2, 0, P4ticks2);
217     pwm1.setPWM(3, 0, P4ticks3);
218     // PATA 5
219     pwm1.setPWM(4, 0, P5ticks1);
220     pwm1.setPWM(5, 0, P5ticks2);
221     pwm1.setPWM(6, 0, P5ticks3);
222     // PATA 6
223     pwm1.setPWM(7, 0, P6ticks1);
224     pwm1.setPWM(8, 0, P6ticks2);
```

```
225     pwm1.setPWM(9, 0, P6ticks3);
226   }
227
228   void loop() {
229     // Limpiamos el buffer del BT
230     while (2 < (SerialBT.available())) {
231       SerialBT.read();
232     }
233     // Lectura del buffer BT y guardado del valor leído
234     if (SerialBT.available()) {
235       dato = SerialBT.read();
236       if ((dato == 'N') || (dato == 'U') || (dato == 'O')) {
237         cambiomodo(dato);
238         enviaticks(matrixRP[0][0], matrixRP[0][1], matrixRP[0][2],
239                    matrixRP[0][3], matrixRP[0][4], matrixRP[0][5],
240                    matrixRP[0][6], matrixRP[0][7], matrixRP[0][8],
241                    matrixRP[0][9], matrixRP[0][10],
242   matrixRP[0][11],
243                    matrixRP[0][12], matrixRP[0][13],
244   matrixRP[0][14],
245                    matrixRP[0][15], matrixRP[0][16],
246   matrixRP[0][17]);
247       }
248     }
249
250     // Volver a Reposo desde FORWARD
251     if ((dato == 'A') && (previo == 'F')) {
252       previo = dato;
253       for (int i = 0; i < rowsA; i++) {
254         P1ticks1 = matrixRN[i][0];
255         P1ticks2 = matrixRN[i][1];
256         P1ticks3 = matrixRN[i][2];
257         P2ticks1 = matrixRN[i][3];
258         P2ticks2 = matrixRN[i][4];
259         P2ticks3 = matrixRN[i][5];
260         P3ticks1 = matrixRN[i][6];
261         P3ticks2 = matrixRN[i][7];
262         P3ticks3 = matrixRN[i][8];
263         P4ticks1 = matrixRN[i][9];
264         P4ticks2 = matrixRN[i][10];
265         P4ticks3 = matrixRN[i][11];
266         P5ticks1 = matrixRN[i][12];
267         P5ticks2 = matrixRN[i][13];
268         P5ticks3 = matrixRN[i][14];
269         P6ticks1 = matrixRN[i][15];
270         P6ticks2 = matrixRN[i][16];
271         P6ticks3 = matrixRN[i][17];
272
273         enviaticks(P1ticks1, P1ticks2, P1ticks3,
274                    P2ticks1, P2ticks2, P2ticks3,
275                    P3ticks1, P3ticks2, P3ticks3,
276                    P4ticks1, P4ticks2, P4ticks3,
277                    P5ticks1, P5ticks2, P5ticks3,
278                    P6ticks1, P6ticks2, P6ticks3);
279       }
280     }
281
```

```
282     // Volver a Reposo desde BACKWARD
283     if ((dato == 'A') && (previo == 'D')) {
284       previo = dato;
285       for (int i = rowsA - 1; i >= 0; i--) {
286         P1ticks1 = matrixRP[i][0];
287         P1ticks2 = matrixRP[i][1];
288         P1ticks3 = matrixRP[i][2];
289         P2ticks1 = matrixRP[i][3];
290         P2ticks2 = matrixRP[i][4];
291         P2ticks3 = matrixRP[i][5];
292         P3ticks1 = matrixRP[i][6];
293         P3ticks2 = matrixRP[i][7];
294         P3ticks3 = matrixRP[i][8];
295         P4ticks1 = matrixRP[i][9];
296         P4ticks2 = matrixRP[i][10];
297         P4ticks3 = matrixRP[i][11];
298         P5ticks1 = matrixRP[i][12];
299         P5ticks2 = matrixRP[i][13];
300         P5ticks3 = matrixRP[i][14];
301         P6ticks1 = matrixRP[i][15];
302         P6ticks2 = matrixRP[i][16];
303         P6ticks3 = matrixRP[i][17];
304
305         enviaticks(P1ticks1, P1ticks2, P1ticks3,
306                    P2ticks1, P2ticks2, P2ticks3,
307                    P3ticks1, P3ticks2, P3ticks3,
308                    P4ticks1, P4ticks2, P4ticks3,
309                    P5ticks1, P5ticks2, P5ticks3,
310                    P6ticks1, P6ticks2, P6ticks3);
311       }
312     }
313
314     // Volver a Reposo desde RIGHT
315     if ((dato == 'A') && (previo == 'R')) {
316       previo = dato;
317       for (int i = 0; i < rowsA; i++) {
318         P1ticks1 = matrixROTRN[i][0];
319         P1ticks2 = matrixROTRN[i][1];
320         P1ticks3 = matrixROTRN[i][2];
321         P2ticks1 = matrixROTRN[i][3];
322         P2ticks2 = matrixROTRN[i][4];
323         P2ticks3 = matrixROTRN[i][5];
324         P3ticks1 = matrixROTRN[i][6];
325         P3ticks2 = matrixROTRN[i][7];
326         P3ticks3 = matrixROTRN[i][8];
327         P4ticks1 = matrixROTRN[i][9];
328         P4ticks2 = matrixROTRN[i][10];
329         P4ticks3 = matrixROTRN[i][11];
330         P5ticks1 = matrixROTRN[i][12];
331         P5ticks2 = matrixROTRN[i][13];
332         P5ticks3 = matrixROTRN[i][14];
333         P6ticks1 = matrixROTRN[i][15];
334         P6ticks2 = matrixROTRN[i][16];
335         P6ticks3 = matrixROTRN[i][17];
336
337         enviaticks(P1ticks1, P1ticks2, P1ticks3,
338                    P2ticks1, P2ticks2, P2ticks3,
```

```
339                   P3ticks1, P3ticks2, P3ticks3,
340                   P4ticks1, P4ticks2, P4ticks3,
341                   P5ticks1, P5ticks2, P5ticks3,
342                   P6ticks1, P6ticks2, P6ticks3);
343       }
344     }
345
346     // Volver a Reposo desde LEFT
347     if ((dato == 'A') && (previo == 'L')) {
348       previo = dato;
349       for (int i = rowsA - 1; i >= 0; i--) {
350         P1ticks1 = matrixROTRP[i][0];
351         P1ticks2 = matrixROTRP[i][1];
352         P1ticks3 = matrixROTRP[i][2];
353         P2ticks1 = matrixROTRP[i][3];
354         P2ticks2 = matrixROTRP[i][4];
355         P2ticks3 = matrixROTRP[i][5];
356         P3ticks1 = matrixROTRP[i][6];
357         P3ticks2 = matrixROTRP[i][7];
358         P3ticks3 = matrixROTRP[i][8];
359         P4ticks1 = matrixROTRP[i][9];
360         P4ticks2 = matrixROTRP[i][10];
361         P4ticks3 = matrixROTRP[i][11];
362         P5ticks1 = matrixROTRP[i][12];
363         P5ticks2 = matrixROTRP[i][13];
364         P5ticks3 = matrixROTRP[i][14];
365         P6ticks1 = matrixROTRP[i][15];
366         P6ticks2 = matrixROTRP[i][16];
367         P6ticks3 = matrixROTRP[i][17];
368
369         enviaticks(P1ticks1, P1ticks2, P1ticks3,
370                    P2ticks1, P2ticks2, P2ticks3,
371                    P3ticks1, P3ticks2, P3ticks3,
372                    P4ticks1, P4ticks2, P4ticks3,
373                    P5ticks1, P5ticks2, P5ticks3,
374                    P6ticks1, P6ticks2, P6ticks3);
375       }
376     }
377
378     // PASO FORWARD
379     if (dato == 'F') {
380       if (previo == 'A') {
381         for (int i = 0; i < rowsA; i++) {
382           P1ticks1 = matrixRP[i][0];
383           P1ticks2 = matrixRP[i][1];
384           P1ticks3 = matrixRP[i][2];
385           P2ticks1 = matrixRP[i][3];
386           P2ticks2 = matrixRP[i][4];
387           P2ticks3 = matrixRP[i][5];
388           P3ticks1 = matrixRP[i][6];
389           P3ticks2 = matrixRP[i][7];
390           P3ticks3 = matrixRP[i][8];
391           P4ticks1 = matrixRP[i][9];
392           P4ticks2 = matrixRP[i][10];
393           P4ticks3 = matrixRP[i][11];
394           P5ticks1 = matrixRP[i][12];
395           P5ticks2 = matrixRP[i][13];
```

```
396           P5ticks3 = matrixRP[i][14];
397           P6ticks1 = matrixRP[i][15];
398           P6ticks2 = matrixRP[i][16];
399           P6ticks3 = matrixRP[i][17];
400
401           enviaticks(P1ticks1, P1ticks2, P1ticks3,
402                      P2ticks1, P2ticks2, P2ticks3,
403                      P3ticks1, P3ticks2, P3ticks3,
404                      P4ticks1, P4ticks2, P4ticks3,
405                      P5ticks1, P5ticks2, P5ticks3,
406                      P6ticks1, P6ticks2, P6ticks3);
407         }
408       }
409       previo = dato;
410       for (int i = 0; i < rows; i++) {
411         P1ticks1 = matrixFW[i][0];
412         P1ticks2 = matrixFW[i][1];
413         P1ticks3 = matrixFW[i][2];
414         P2ticks1 = matrixFW[i][3];
415         P2ticks2 = matrixFW[i][4];
416         P2ticks3 = matrixFW[i][5];
417         P3ticks1 = matrixFW[i][6];
418         P3ticks2 = matrixFW[i][7];
419         P3ticks3 = matrixFW[i][8];
420         P4ticks1 = matrixFW[i][9];
421         P4ticks2 = matrixFW[i][10];
422         P4ticks3 = matrixFW[i][11];
423         P5ticks1 = matrixFW[i][12];
424         P5ticks2 = matrixFW[i][13];
425         P5ticks3 = matrixFW[i][14];
426         P6ticks1 = matrixFW[i][15];
427         P6ticks2 = matrixFW[i][16];
428         P6ticks3 = matrixFW[i][17];
429
430         enviaticks(P1ticks1, P1ticks2, P1ticks3,
431                    P2ticks1, P2ticks2, P2ticks3,
432                    P3ticks1, P3ticks2, P3ticks3,
433                    P4ticks1, P4ticks2, P4ticks3,
434                    P5ticks1, P5ticks2, P5ticks3,
435                    P6ticks1, P6ticks2, P6ticks3);
436       }
437     }
438
439     // PASO BACKWARD
440     if (dato == 'D') {
441       if (previo == 'A') {
442         for (int i = rowsA - 1; i >= 0; i--) {
443           P1ticks1 = matrixRN[i][0];
444           P1ticks2 = matrixRN[i][1];
445           P1ticks3 = matrixRN[i][2];
446           P2ticks1 = matrixRN[i][3];
447           P2ticks2 = matrixRN[i][4];
448           P2ticks3 = matrixRN[i][5];
449           P3ticks1 = matrixRN[i][6];
450           P3ticks2 = matrixRN[i][7];
451           P3ticks3 = matrixRN[i][8];
452           P4ticks1 = matrixRN[i][9];
```

```
453           P4ticks2 = matrixRN[i][10];
454           P4ticks3 = matrixRN[i][11];
455           P5ticks1 = matrixRN[i][12];
456           P5ticks2 = matrixRN[i][13];
457           P5ticks3 = matrixRN[i][14];
458           P6ticks1 = matrixRN[i][15];
459           P6ticks2 = matrixRN[i][16];
460           P6ticks3 = matrixRN[i][17];
461
462           enviaticks(P1ticks1, P1ticks2, P1ticks3,
463                      P2ticks1, P2ticks2, P2ticks3,
464                      P3ticks1, P3ticks2, P3ticks3,
465                      P4ticks1, P4ticks2, P4ticks3,
466                      P5ticks1, P5ticks2, P5ticks3,
467                      P6ticks1, P6ticks2, P6ticks3);
468         }
469       }
470       previo = dato;
471       for (int i = rows; i >= 0; i--) {
472         P1ticks1 = matrixFW[i][0];
473         P1ticks2 = matrixFW[i][1];
474         P1ticks3 = matrixFW[i][2];
475         P2ticks1 = matrixFW[i][3];
476         P2ticks2 = matrixFW[i][4];
477         P2ticks3 = matrixFW[i][5];
478         P3ticks1 = matrixFW[i][6];
479         P3ticks2 = matrixFW[i][7];
480         P3ticks3 = matrixFW[i][8];
481         P4ticks1 = matrixFW[i][9];
482         P4ticks2 = matrixFW[i][10];
483         P4ticks3 = matrixFW[i][11];
484         P5ticks1 = matrixFW[i][12];
485         P5ticks2 = matrixFW[i][13];
486         P5ticks3 = matrixFW[i][14];
487         P6ticks1 = matrixFW[i][15];
488         P6ticks2 = matrixFW[i][16];
489         P6ticks3 = matrixFW[i][17];
490
491         enviaticks(P1ticks1, P1ticks2, P1ticks3,
492                    P2ticks1, P2ticks2, P2ticks3,
493                    P3ticks1, P3ticks2, P3ticks3,
494                    P4ticks1, P4ticks2, P4ticks3,
495                    P5ticks1, P5ticks2, P5ticks3,
496                    P6ticks1, P6ticks2, P6ticks3);
497       }
498     }
499
500     // GIRO RIGHT
501     if (dato == 'R') {
502       if (previo == 'A') {
503         for (int i = 0; i < rowsA; i++) {
504           P1ticks1 = matrixROTRP[i][0];
505           P1ticks2 = matrixROTRP[i][1];
506           P1ticks3 = matrixROTRP[i][2];
507           P2ticks1 = matrixROTRP[i][3];
508           P2ticks2 = matrixROTRP[i][4];
509           P2ticks3 = matrixROTRP[i][5];
```

```
510           P3ticks1 = matrixROTRP[i][6];
511           P3ticks2 = matrixROTRP[i][7];
512           P3ticks3 = matrixROTRP[i][8];
513           P4ticks1 = matrixROTRP[i][9];
514           P4ticks2 = matrixROTRP[i][10];
515           P4ticks3 = matrixROTRP[i][11];
516           P5ticks1 = matrixROTRP[i][12];
517           P5ticks2 = matrixROTRP[i][13];
518           P5ticks3 = matrixROTRP[i][14];
519           P6ticks1 = matrixROTRP[i][15];
520           P6ticks2 = matrixROTRP[i][16];
521           P6ticks3 = matrixROTRP[i][17];
522
523           enviaticks(P1ticks1, P1ticks2, P1ticks3,
524                      P2ticks1, P2ticks2, P2ticks3,
525                      P3ticks1, P3ticks2, P3ticks3,
526                      P4ticks1, P4ticks2, P4ticks3,
527                      P5ticks1, P5ticks2, P5ticks3,
528                      P6ticks1, P6ticks2, P6ticks3);
529         }
530       }
531       previo = dato;
532       for (int i = 0; i < rows; i++) {
533         P1ticks1 = matrixRT[i][0];
534         P1ticks2 = matrixRT[i][1];
535         P1ticks3 = matrixRT[i][2];
536         P2ticks1 = matrixRT[i][3];
537         P2ticks2 = matrixRT[i][4];
538         P2ticks3 = matrixRT[i][5];
539         P3ticks1 = matrixRT[i][6];
540         P3ticks2 = matrixRT[i][7];
541         P3ticks3 = matrixRT[i][8];
542         P4ticks1 = matrixRT[i][9];
543         P4ticks2 = matrixRT[i][10];
544         P4ticks3 = matrixRT[i][11];
545         P5ticks1 = matrixRT[i][12];
546         P5ticks2 = matrixRT[i][13];
547         P5ticks3 = matrixRT[i][14];
548         P6ticks1 = matrixRT[i][15];
549         P6ticks2 = matrixRT[i][16];
550         P6ticks3 = matrixRT[i][17];
551
552         enviaticks(P1ticks1, P1ticks2, P1ticks3,
553                    P2ticks1, P2ticks2, P2ticks3,
554                    P3ticks1, P3ticks2, P3ticks3,
555                    P4ticks1, P4ticks2, P4ticks3,
556                    P5ticks1, P5ticks2, P5ticks3,
557                    P6ticks1, P6ticks2, P6ticks3);
558       }
559     }
560
561     // GIRO LEFT
562     if (dato == 'L') {
563       if (previo == 'A') {
564         for (int i = rowsA - 1; i >= 0; i--) {
565           P1ticks1 = matrixROTRN[i][0];
566           P1ticks2 = matrixROTRN[i][1];
```

```
567           P1ticks3 = matrixROTRN[i][2];
568           P2ticks1 = matrixROTRN[i][3];
569           P2ticks2 = matrixROTRN[i][4];
570           P2ticks3 = matrixROTRN[i][5];
571           P3ticks1 = matrixROTRN[i][6];
572           P3ticks2 = matrixROTRN[i][7];
573           P3ticks3 = matrixROTRN[i][8];
574           P4ticks1 = matrixROTRN[i][9];
575           P4ticks2 = matrixROTRN[i][10];
576           P4ticks3 = matrixROTRN[i][11];
577           P5ticks1 = matrixROTRN[i][12];
578           P5ticks2 = matrixROTRN[i][13];
579           P5ticks3 = matrixROTRN[i][14];
580           P6ticks1 = matrixROTRN[i][15];
581           P6ticks2 = matrixROTRN[i][16];
582           P6ticks3 = matrixROTRN[i][17];
583
584           enviaticks(P1ticks1, P1ticks2, P1ticks3,
585                      P2ticks1, P2ticks2, P2ticks3,
586                      P3ticks1, P3ticks2, P3ticks3,
587                      P4ticks1, P4ticks2, P4ticks3,
588                      P5ticks1, P5ticks2, P5ticks3,
589                      P6ticks1, P6ticks2, P6ticks3);
590         }
591       }
592       previo = dato;
593       for (int i = rows; i >= 0; i--) {
594         P1ticks1 = matrixRT[i][0];
595         P1ticks2 = matrixRT[i][1];
596         P1ticks3 = matrixRT[i][2];
597         P2ticks1 = matrixRT[i][3];
598         P2ticks2 = matrixRT[i][4];
599         P2ticks3 = matrixRT[i][5];
600         P3ticks1 = matrixRT[i][6];
601         P3ticks2 = matrixRT[i][7];
602         P3ticks3 = matrixRT[i][8];
603         P4ticks1 = matrixRT[i][9];
604         P4ticks2 = matrixRT[i][10];
605         P4ticks3 = matrixRT[i][11];
606         P5ticks1 = matrixRT[i][12];
607         P5ticks2 = matrixRT[i][13];
608         P5ticks3 = matrixRT[i][14];
609         P6ticks1 = matrixRT[i][15];
610         P6ticks2 = matrixRT[i][16];
611         P6ticks3 = matrixRT[i][17];
612
613         enviaticks(P1ticks1, P1ticks2, P1ticks3,
614                    P2ticks1, P2ticks2, P2ticks3,
615                    P3ticks1, P3ticks2, P3ticks3,
616                    P4ticks1, P4ticks2, P4ticks3,
617                    P5ticks1, P5ticks2, P5ticks3,
618                    P6ticks1, P6ticks2, P6ticks3);
619       }
620     }
621   }
```

10.8.11 cd_hexapod.m

```
function [x3, y3, z3] = cd(theta1, theta2, theta3, l1, l2, l3)

  function t = T(d, th, a, al) % Calculo de la matriz de
transformacion
    t = [cos(th), -cos(al)*sin(th), sin(al)*sin(th), a*cos(th);
         sin(th), cos(al)*cos(th), -sin(al)*cos(th), a*sin(th);
         0, sin(al), cos(al), d;
         0, 0, 0, 1];
  end

  d = [0, 0, 0];
  theta = [theta1, theta2, theta3];
  a = [l1, l2, l3];
  al = [pi/2, 0, 0];

  O00 = [0, 0, 0, 1]';
  O11 = [0, 0, 0, 1]';
  O22 = [0, 0, 0, 1]';
  O33 = [0, 0, 0, 1]';

  T01 = T(d(1), theta(1), a(1), al(1));
  T12 = T(d(2), theta(2), a(2), al(2));
  T23 = T(d(3), theta(3), a(3), al(3));

  T02 = T01 * T12;
  T03 = T02 * T23;

  O10 = T01 * O11;
  O20 = T02 * O22;
  O30 = T03 * O33;

  x3 = O30(1);
  y3 = O30(2);
  z3 = O30(3);

  m = [O00, O10, O20, O30];
  plot3(m(1, :), m(2, :), m(3, :), 'LineWidth', 2); % Links más
gruesos
  hold on;
  plot3(m(1, :), m(2, :), m(3, :), 'r.', 'MarkerSize', 25); %
Puntos más grandes

  % Dibujar las líneas discontinuas en las posiciones correctas
  % Línea desde el origen en la dirección del eje X
  plot3([O00(1), O00(1) + l1], [O00(2), O00(2)], ...
        [O00(3), O00(3)], 'k--', 'LineWidth', 1);
  % Línea desde la primera articulación en la dirección del
primer link
  plot3([O10(1), O10(1) + l1*cos(theta1)], ...
        [O10(2), O10(2) + l1*sin(theta1)], ...
        [O10(3), O10(3)], 'k--', 'LineWidth', 1);
  % Línea desde la segunda articulación en la dirección del
segundo link
```

```
54    dir_link2 = T02 * [l2; 0; 0; 0];
55    plot3([O20(1), O20(1) + 2.2*dir_link2(1)], ...
56          [O20(2), O20(2) + 2.2*dir_link2(2)], ...
57          [O20(3), O20(3) + 2.2*dir_link2(3)], ...
58          'k--', 'LineWidth', 1);
59
60    % Dibujar los arcos para los ángulos
61    % Arco para theta1
62    angle = linspace(0, theta1, 100);
63    x_arc = l1 * cos(angle);
64    y_arc = l1 * sin(angle);
65    z_arc = zeros(size(angle));
66    plot3(x_arc, y_arc, z_arc, 'b', 'LineWidth', 1);
67
68    % Arco para theta2
69    angle = linspace(0, theta2, 100);
70    x_arc = O10(1) + l2 * cos(angle) * cos(theta1);
71    y_arc = O10(2) + l2 * cos(angle) * sin(theta1);
72    z_arc = O10(3) + l2 * sin(angle);
73    plot3(x_arc, y_arc, z_arc, 'g', 'LineWidth', 1); % Verde oscuro
74
75    % Arco para theta3
76    angle = linspace(0, theta3, 100);
77    x_arc = zeros(size(angle));
78    y_arc = zeros(size(angle));
79    z_arc = zeros(size(angle));
80    for i = 1:length(angle)
81      T_full = T01 * T12 * T(d(3), angle(i), l3, al(3));
82      O = T_full * [0; 0; 0; 1];
83      x_arc(i) = O(1);
84      y_arc(i) = O(2);
85      z_arc(i) = O(3);
86    end
87    plot3(x_arc, y_arc, z_arc, 'm', 'LineWidth', 1);
88
89    ejes = axis(); % ejes es un vector de tipo [xmin xmax ymin ymax
90  zmin zmax]
91    largo = [0, max([ejes(2) - ejes(1), ejes(4) - ejes(3), ejes(6)
92  - ejes(5)])];
93    axis([ejes(1) + largo, ejes(3) + largo, ejes(5) + largo]);
94
95    xlabel('X', 'FontSize', 14);
96    ylabel('Y', 'FontSize', 14);
97    zlabel('Z', 'FontSize', 14);
98    grid();
99    title('3D', 'FontSize', 16);
100
101   % Calcular las posiciones de las etiquetas en la mitad de cada
102 link
103   mid_link1 = (O00 + O10) / 2;
104   mid_link2 = (O10 + O20) / 2;
105   mid_link3 = (O20 + O30) / 2;
106
107   % Añadir etiquetas a los links
108   text(mid_link1(1), mid_link1(2), mid_link1(3), 'COXA', ...
109        'FontSize', 20, 'VerticalAlignment', 'bottom', ...
110        'HorizontalAlignment', 'right');
```

```
111     text(mid_link2(1), mid_link2(2), mid_link2(3), 'FEMUR', ...
112          'FontSize', 20, 'VerticalAlignment', 'bottom', ...
113          'HorizontalAlignment', 'right');
114     text(mid_link3(1), mid_link3(2), mid_link3(3), 'TIBIA', ...
115          'FontSize', 20, 'VerticalAlignment', 'bottom', ...
116          'HorizontalAlignment', 'right');
117 
118     % Añadir texto con las coordenadas del punto final
119     coords_text = sprintf('(%0.2f, %0.2f, %0.2f)', x3, y3, z3);
120     text(x3, y3, z3, coords_text, 'FontSize', 20,
121 'VerticalAlignment', ...
122          'top', 'HorizontalAlignment', 'left');
123 
124     % Añadir etiquetas con los ángulos en las posiciones de las
125 articulaciones
126     text(O00(1), O00(2), O00(3), sprintf('\\theta_1 = %0.2f',
127 rad2deg(theta1)), ...
128          'FontSize', 18, 'Color', 'b', 'VerticalAlignment', 'top',
129 ...
130          'HorizontalAlignment', 'right');
131     text(O10(1), O10(2), O10(3), sprintf('\\theta_2 = %0.2f',
132 rad2deg(theta2)), ...
133          'FontSize', 18, 'Color', 'g', 'VerticalAlignment', 'top',
134 ...
135          'HorizontalAlignment', 'right'); % Verde oscuro
136     text(O20(1), O20(2), O20(3), sprintf('\\theta_3 = %0.2f',
137 rad2deg(theta3)), ...
138          'FontSize', 18, 'Color', 'm', 'VerticalAlignment', 'top',
139 ...
140          'HorizontalAlignment', 'right');
141 
142     hold off;
143 end
```

10.8.12 ci_hexapod.m

```
function [theta1, theta2, theta3] = ci_hexapod(x, y, z)
  % Definición de longitudes
  l1 = 55.975; %81.998
  l2 = 81.311; %86.104
  l3 = 155.185; %188.858
  codo = 1;

  % Calculo de theta1
  theta1 = atan2(y, x);
  x1 = x - l1 * cos(theta1);
  y1 = y - l1 * sin(theta1);

  % Calculo de cos(theta3)
  c3 = (x1^2 + y1^2 + z^2 - l2^2 - l3^2) / (2 * l2 * l3);

  % Verificación de si el punto es alcanzable
  if(abs(c3) <= 1)
      disp('El punto es alcanzable');
  else
      disp('El punto NO es alcanzable');
      theta1 = NaN;
      theta2 = NaN;
      theta3 = NaN;
      return;
  end

  % CODO ARRIBA O ABAJO
  if(codo == 0)  % CODO ABAJO
      theta3 = acos(c3);
  elseif(codo == 1)  % CODO ARRIBA
      theta3 = -acos(c3);
  end

  % Calculo de theta2
  phi = atan2(z, sqrt(x1^2 + y1^2));
  alfa = atan2(l3 * sin(theta3), l2 + l3 * cos(theta3));
  theta2 = phi - alfa;

  % Llamada a la función cd_hexapod
  cd_hexapod(theta1, theta2, theta3, l1, l2, l3);

  % Mostrar resultados
  disp(['theta1: ', num2str(theta1)]);
  disp(['theta2: ', num2str(theta2)]);
  disp(['theta3: ', num2str(theta3)]);
end
```

10.9 Hojas de datos

Se muestran las hojas de datos del ESP32 y de los módulos PWM PCA9685.

ESP32 Datasheet

Espressif Systems

October 8, 2016

About This Guide

This document provides introduction to the specifications of ESP32 hardware.

The document structure is as follows:

Chapter	Title	Subject
Chapter 1	Overview	An overview of ESP32, including featured solutions, basic and advanced features, applications and development support
Chapter 2	Pin Definitions	Introduction to the pin layout and descriptions
Chapter 3	Functional Description	Description of the major functional modules
Chapter 4	Peripheral Interface	Description of the peripheral interfaces integrated on ESP32
Chapter 5	Electrical Characteristics	The electrical characteristics and data of ESP32
Chapter 6	Package Information	The package details of ESP32
Chapter 7	Supported Resources	The related documents and community resources for ESP32
Appendix	Touch Sensor	The touch sensor design and layout guidelines

Release Notes

Date	Version	Release notes
2016.08	V1.0	First release

Disclaimer and Copyright Notice

Contents

List of Tables

List of Figures

1. Overview

ESP32 is a single chip 2.4 GHz Wi-Fi and Bluetooth combo chip designed with TSMC ultra low power 40 nm technology. It is designed and optimized for the best power performance, RF performance, robustness, versatility, features and reliability, for a wide variety of applications, and different power profiles.

1.1 Featured Solutions

1.1.1 Ultra Low Power Solution

ESP32 is designed for mobile, wearable electronics, and Internet of Things (IoT) applications. It has many features of the state-of-the-art low power chips, including fine resolution clock gating, power modes, and dynamic power scaling.

For instance, in a low-power IoT sensor hub application scenario, ESP32 is woken up periodically and only when a specified condition is detected; low duty cycle is used to minimize the amount of energy that the chip expends. The output power of the power amplifier is also adjustable to achieve an optimal trade off between communication range, data rate and power consumption.

Note:
For more information, refer to Section 3.7 RTC and Low-Power Management.

1.1.2 Complete Integration Solution

ESP32 is the most integrated solution for Wi-Fi + Bluetooth applications in the industry with less than 10 external components. ESP32 integrates the antenna switch, RF balun, power amplifier, low noise receive amplifier, filters, and power management modules. As such, the entire solution occupies minimal Printed Circuit Board (PCB) area.

ESP32 uses CMOS for single-chip fully-integrated radio and baseband, and also integrates advanced calibration circuitries that allow the solution to dynamically adjust itself to remove external circuit imperfections or adjust to changes in external conditions.

As such, the mass production of ESP32 solutions does not require expensive and specialized Wi-Fi test equipment.

1.2 Basic Protocols

1.2.1 Wi-Fi

- 802.11 b/g/n/e/i
- 802.11 n (2.4 GHz), up to 150 Mbps
- 802.11 e: QoS for wireless multimedia technology
- WMM-PS, UAPSD
- A-MPDU and A-MSDU aggregation
- Block ACK

- Fragmentation and defragmentation
- Automatic Beacon monitoring/scanning
- 802.11 i security features: pre-authentication and TSN
- Wi-Fi Protected Access (WPA)/WPA2/WPA2-Enterprise/Wi-Fi Protected Setup (WPS)
- Infrastructure BSS Station mode/SoftAP mode
- Wi-Fi Direct (P2P), P2P Discovery, P2P Group Owner mode and P2P Power Management
- UMA compliant and certified
- Antenna diversity and selection

Note:
For more information, refer to Section 3.5 Wi-Fi.

1.2.2 Bluetooth

- Compliant with Bluetooth v4.2 BR/EDR and BLE specification
- Class-1, class-2 and class-3 transmitter without external power amplifier
- Enhanced power control
- +10 dBm transmitting power
- NZIF receiver with -98 dBm sensitivity
- Adaptive Frequency Hopping (AFH)
- Standard HCI based on SDIO/SPI/UART
- High speed UART HCI, up to 4 Mbps
- BT 4.2 controller and host stack
- Service Discover Protocol (SDP)
- General Access Profile (GAP)
- Security Manage Protocol (SMP)
- Bluetooth Low Energy (BLE)
- ATT/GATT
- HID
- All GATT-based profile supported
- SPP-Like GATT-based profile
- BLE Beacon
- A2DP/AVRCP/SPP, HSP/HFP, RFCOMM
- CVSD and SBC for audio codec
- Bluetooth Piconet and Scatternet

1.3 MCU and Advanced Features

1.3.1 CPU and Memory

- Xtensa® Dual-Core 32-bit LX6 microprocessors, up to 600 DMIPS
- 448 KByte ROM
- 520 KByte SRAM
- 16 KByte SRAM in RTC
- QSPI Flash/SRAM, up to 4 x 16 MBytes
- Power supply: 2.2 V to 3.6 V

1.3.2 Clocks and Timers

- Internal 8 MHz oscillator with calibration
- Internal RC oscillator with calibration
- External 2 MHz to 40 MHz crystal oscillator
- External 32 kHz crystal oscillator for RTC with calibration
- Two timer groups, including 2 x 64-bit timers and 1 x main watchdog in each group
- RTC timer with sub-second accuracy
- RTC watchdog

1.3.3 Advanced Peripheral Interfaces

- 12-bit SAR ADC up to 18 channels
- 2 × 8-bit D/A converters
- 10 × touch sensors
- Temperature sensor
- 4 × SPI
- 2 × I2S
- 2 × I2C
- 3 × UART
- 1 host (SD/eMMC/SDIO)
- 1 slave (SDIO/SPI)
- Ethernet MAC interface with dedicated DMA and IEEE 1588 support
- CAN 2.0
- IR (TX/RX)
- Motor PWM
- LED PWM up to 16 channels
- Hall sensor
- Ultra low power analog pre-amplifier

1.3.4 Security

- IEEE 802.11 standard security features all supported, including WFA, WPA/WPA2 and WAPI
- Secure boot
- Flash encryption
- 1024-bit OTP, up to 768-bit for customers
- Cryptographic hardware acceleration:
  - AES
  - HASH (SHA-2) library
  - RSA
  - ECC
  - Random Number Generator (RNG)

1.3.5 Development Support

- SDK Firmware for fast on-line programming
- Open source toolchains based on GCC

Note:
For more information, refer to Chapter 7 Supported Resources.

1.4 Application

- Generic low power IoT sensor hub
- Generic low power IoT loggers
- Video streaming from camera
- Over The Top (OTT) devices
- Music players
  - Internet music players
  - Audio streaming devices
- Wi-Fi enabled toys
  - Loggers
  - Proximity sensing toys
- Wi-Fi enabled speech recognition devices
- Audio headsets
- Smart power plugs
- Home automation
- Mesh network

- Industrial wireless control
- Baby monitors
- Wearable electronics
- Wi-Fi location-aware devices
- Security ID tags
- Healthcare
  - Proximity and movement monitoring trigger devices
  - Temperature sensing loggers

1.5 Block Diagram

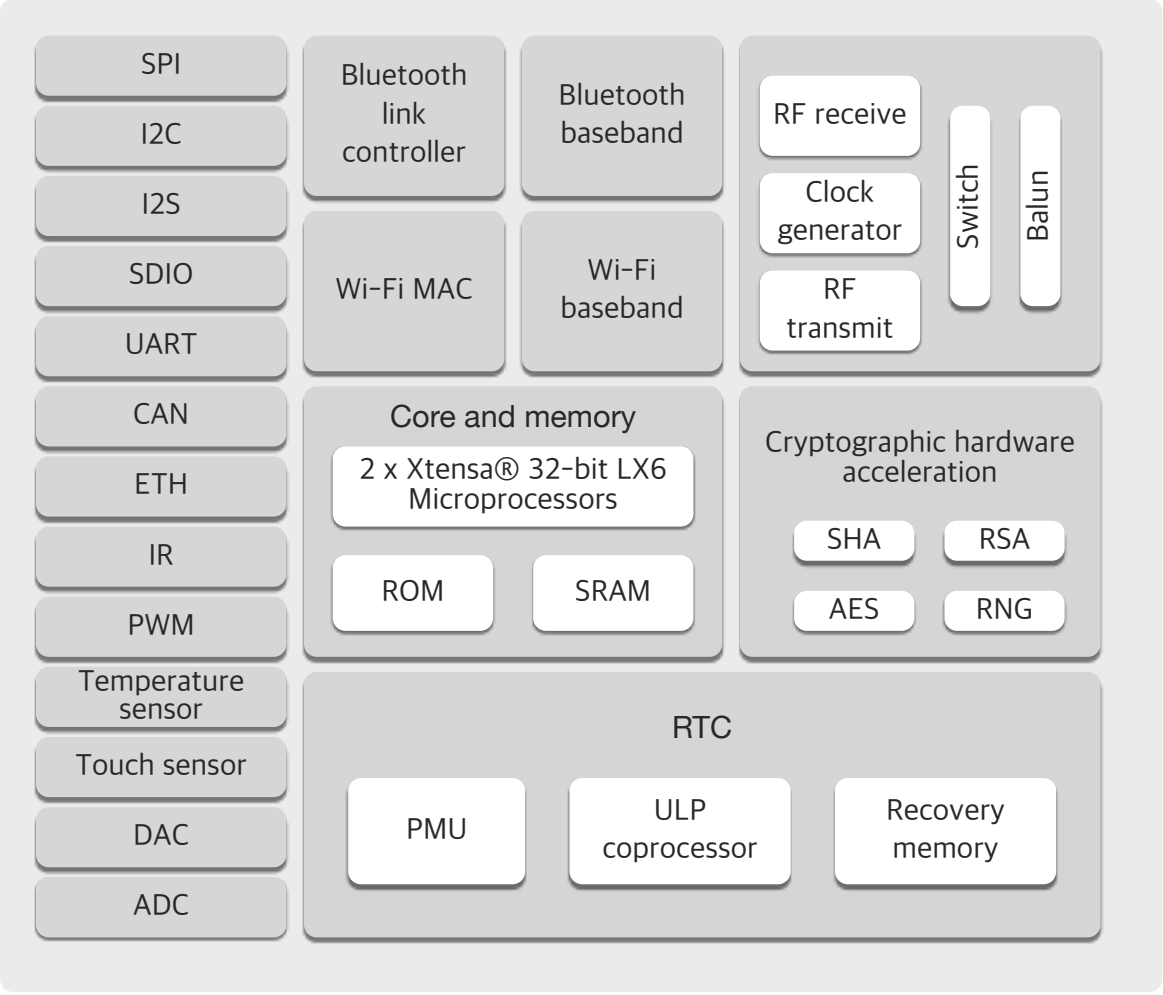


Figure 1: Function Block Diagram

2. Pin Definitions

2.1 Pin Layout

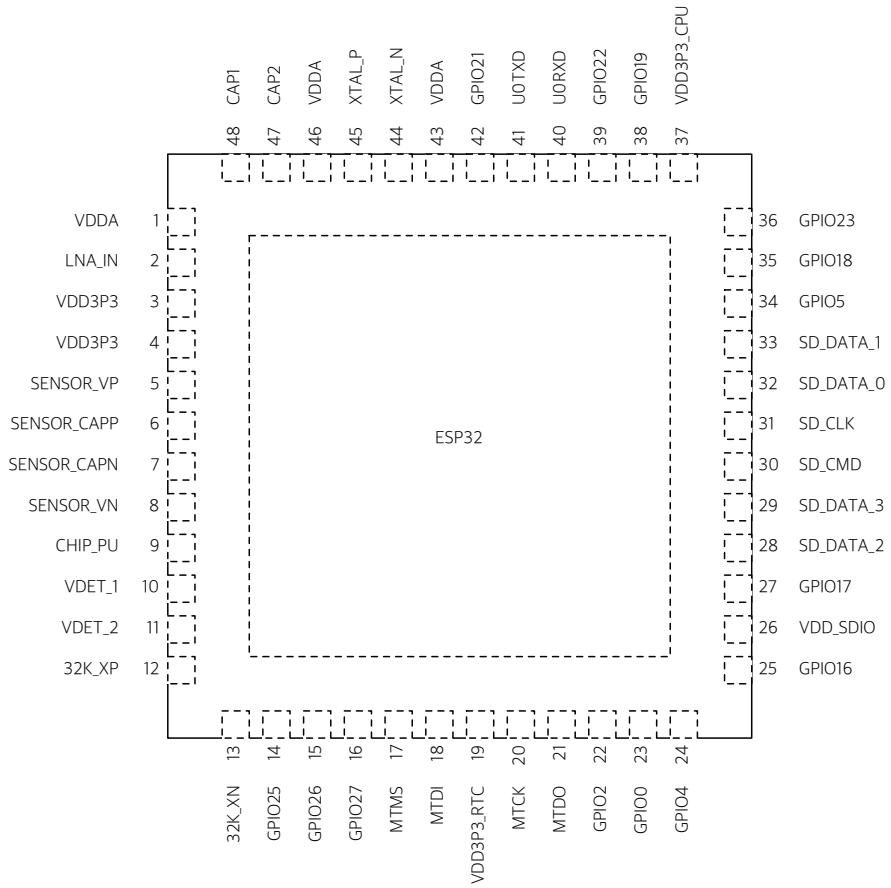


Figure 2: ESP32 Pin Layout

2.2 Pin Description

Table 1: Pin Description

Name	No.	Type	Function
Analog			
VDDA	1	P	Analog power supply (2.3V ~ 3.6V)
LNA_IN	2	I/O	RF input and output
VDD3P3	3	P	Amplifier power supply (2.3V ~ 3.6V)
VDD3P3	4	P	Amplifier power supply (2.3V ~ 3.6V)
VDD3P3_RTC			
SENSOR_VP	5	I	GPIO36, ADC_PRE_AMP, ADC1_CH0, RTC_GPIO0 Note: Connects 270 pF capacitor from SENSOR_VP to SENSOR_CAPP when used as ADC_PRE_AMP.

Name	No.	Type	Function
SENSOR_CAPP	6	I	GPIO37, ADC_PRE_AMP, ADC1_CH1, RTC_GPIO1 Note: Connects 270 pF capacitor from SENSOR_VP to SENSOR_CAPP when used as ADC_PRE_AMP.
SENSOR_CAPN	7	I	GPIO38, ADC1_CH2, ADC_PRE_AMP, RTC_GPIO2 Note: Connects 270 pF capacitor from SENSOR_VN to SENSOR_CAPN when used as ADC_PRE_AMP.
SENSOR_VN	8	I	GPIO39, ADC1_CH3, ADC_PRE_AMP, RTC_GPIO3 Note: Connects 270 pF capacitor from SENSOR_VN to SENSOR_CAPN when used as ADC_PRE_AMP.
CHIP_PU	9	I	Chip Enable (Active High) High: On, chip works properly Low: Off, chip works at the minimum power Note: Do not leave CHIP_PU pin floating
VDET_1	10	I	GPIO34, ADC1_CH6, RTC_GPIO4
VDET_2	11	I	GPIO35, ADC1_CH7, RTC_GPIO5
32K_XP	12	I/O	GPIO32, 32K_XP (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
32K_XN	13	I/O	GPIO33, 32K_XN (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
GPIO25	14	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6, EMAC_RXD0
GPIO26	15	I/O	GPIO26, DAC_2, ADC2_CH9, RTC_GPIO7, EMAC_RXD1
GPIO27	16	I/O	GPIO27, ADC2_CH7, TOUCH7, RTC_GPIO17, EMAC_RX_DV
MTMS	17	I/O	GPIO14, ADC2_CH6, TOUCH6, RTC_GPIO16, MTMS, HSPICLK, HS2_CLK, SD_CLK, EMAC_TXD2
MTDI	18	I/O	GPIO12, ADC2_CH5, TOUCH5, RTC_GPIO15, MTDI, HSPIQ, HS2_DATA2, SD_DATA2, EMAC_TXD3
VDD3P3_RTC	19	P	RTC IO power supply input (1.8V - 3.3V)
MTCK	20	I/O	GPIO13, ADC2_CH4, TOUCH4, RTC_GPIO14, MTCK, HSPID, HS2_DATA3, SD_DATA3, EMAC_RX_ER
MTDO	21	I/O	GPIO15, ADC2_CH3, TOUCH3, RTC_GPIO13, MTDO, HSPICS0, HS2_CMD, SD_CMD, EMAC_RXD3
GPIO2	22	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0, SD_DATA0
GPIO0	23	I/O	GPIO0, ADC2_CH1, TOUCH1, RTC_GPIO11, CLK_OUT1, EMAC_TX_CLK
GPIO4	24	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1, SD_DATA1, EMAC_TX_ER
VDD_SDIO			
GPIO16	25	I/O	GPIO16, HS1_DATA4, U2RXD, EMAC_CLK_OUT
VDD_SDIO	26	P	1.8V or 3.3V power supply output
GPIO17	27	I/O	GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180
SD_DATA_2	28	I/O	GPIO9, SD_DATA2, SPIHD, HS1_DATA2, U1RXD
SD_DATA_3	29	I/O	GPIO10, SD_DATA3, SPIWP, HS1_DATA3, U1TXD
SD_CMD	30	I/O	GPIO11, SD_CMD, SPICS0, HS1_CMD, U1RTS
SD_CLK	31	I/O	GPIO6, SD_CLK, SPICLK, HS1_CLK, U1CTS

Name	No.	Type	Function
SD_DATA_0	32	I/O	GPIO7, SD_DATA0, SPIQ, HS1_DATA0, U2RTS
SD_DATA_1	33	I/O	GPIO8, SD_DATA1, SPID, HS1_DATA1, U2CTS
VDD3P3_CPU			
GPIO5	34	I/O	GPIO5, VSPICS0, HS1_DATA6, EMAC_RX_CLK
GPIO18	35	I/O	GPIO18, VSPICLK, HS1_DATA7
GPIO23	36	I/O	GPIO23, VSPID, HS1_STROBE
VDD3P3_CPU	37	P	CPU IO power supply input (1.8V - 3.3V)
GPIO19	38	I/O	GPIO19, VSPIQ, U0CTS, EMAC_TXD0
GPIO22	39	I/O	GPIO22, VSPIWP, U0RTS, EMAC_TXD1
U0RXD	40	I/O	GPIO3, U0RXD, CLK_OUT2
U0TXD	41	I/O	GPIO1, U0TXD, CLK_OUT3, EMAC_RXD2
GPIO21	42	I/O	GPIO21, VSPIHD, EMAC_TX_EN
Analog			
VDDA	43	I/O	Analog power supply (2.3V - 3.6V)
XTAL_N	44	O	External crystal output
XTAL_P	45	I	External crystal input
VDDA	46	P	Digital power supply for PLL (2.3V - 3.6V)
CAP2	47	I	Connects with a 3 nF capacitor and 20 kΩ resistor in parallel to CAP1
CAP1	48	I	Connects with a 10 nF series capacitor to ground

2.3 Power Scheme

ESP32 digital pins are divided into three different power domains:

- VDD3P3_RTC
- VDD3P3_CPU
- VDD_SDIO

VDD3P3_RTC is also the input power supply for RTC and CPU. **VDD3P3_CPU** is also the input power supply for CPU.

VDD_SDIO connects to the output of an internal LDO, whose input is **VDD3P3_RTC**. When **VDD_SDIO** is connected to the same PCB net together with **VDD3P3_RTC**; the internal LDO is disabled automatically.

The internal LDO can be configured as 1.8V, or the same voltage as **VDD3P3_RTC**. It can be powered off via software to minimize the current of Flash/SRAM during the Deep-sleep mode.

Note:
It is required that the power supply of **VDD3P3_RTC**, **VDD3P3_CPU** and analog must be stable before the pin **CHIP_PU** is set at high level.

2.4 Strapping Pins

ESP32 has 6 strapping pins:

- MTDI/GPIO12: internal pull-down
- GPIO0: internal pull-up
- GPIO2: internal pull-down
- GPIO4: internal pull-down
- MTDO/GPIO15: internal pull-up
- GPIO5: internal pull-up

Software can read the value of these 6 bits from the register "GPIO_STRAPPING".

During the chip power-on reset, the latches of the strapping pins sample the voltage level as strapping bits of "0" or "1", and hold these bits until the chip is powered down or shut down. The strapping bits configure the device boot mode, the operating voltage of VDD_SDIO and other system initial settings.

Each strapping pin is connected with its internal pull-up/pull-down during the chip reset. Consequently, if a strapping pin is unconnected or the connected external circuit is high-impendence, the internal weak pull-up/pull-down will determine the default input level of the strapping pins.

To change the strapping bit values, users can apply the external pull-down/pull-up resistances, or apply the host MCU's GPIOs to control the voltage level of these pins when powering on ESP32.

After reset, the strapping pins work as the normal functions pins.

Refer to Table 2 for detailed boot modes configuration by strapping pins.

Table 2: Strapping Pins

Voltage of Internal LDO (VDD_SDIO)					
Pin	Default	3.3V		1.8V	
MTDI	Pull-down	0		1	
Booting Mode					
Pin	Default	SPI Boot		Download Boot	
GPIO0	Pull-up	1		0	
GPIO2	Pull-down	Don't-care		0	
Debugging Log on U0TXD During Booting					
Pin	Default	U0TXD Toggling		U0TXD Silent	
MTDO	Pull-up	1		0	
Timing of SDIO Slave					
Pin	Default	Falling-edge Input Falling-edge Output	Falling-edge Input Rising-edge Output	Rising-edge Input Falling-edge Output	Rising-edge Input Rising-edge Output
MTDO	Pull-up	0	0	1	1
GPIO5	Pull-up	0	1	0	1

Note:
Firmware can configure register bits to change the setting of "Voltage of Internal LDO (VDD_SDIO)" and "Timing of SDIO Slave" after booting.

3. Functional Description

This chapter describes the functions implemented in ESP32.

3.1 CPU and Memory

3.1.1 CPU

ESP32 contains two low-power Xtensa® 32-bit LX6 microprocessors with the following features.

- 7-stage pipeline to support the clock frequency of up to 240 MHz
- 16/24-bit Instruction Set provides high code-density
- Support Floating Point Unit
- Support DSP instructions, such as 32-bit Multiplier, 32-bit Divider, and 40-bit MAC
- Support 32 interrupt vectors from about 70 interrupt sources

The dual CPUs interface through:

- Xtensa RAM/ROM Interface for instruction and data
- Xtensa Local Memory Interface for fast peripheral register access
- Interrupt with external and internal sources
- JTAG interface for debugging

3.1.2 Internal Memory

ESP32's internal memory includes:

- 448 KBytes ROM for booting and core functions
- 520 KBytes on-chip SRAM for data and instruction
- 8 KBytes SRAM in RTC, which is called RTC SLOW Memory and can be used for co-processor accessing during the Deep-sleep mode
- 8 KBytes SRAM in RTC, which is called RTC FAST Memory and can be used for data storage and main CPU during RTC Boot from the Deep-sleep mode
- 1 Kbit of EFUSE, of which 256 bits are used for the system (MAC address and chip configuration) and the remaining 768 bits are reserved for customer applications, including Flash-Encryption and Chip-ID

3.1.3 External Flash and SRAM

ESP32 supports 4 x 16 MBytes of external QSPI Flash and SRAM with hardware encryption based on AES to protect developer's programs and data.

ESP32 accesses external QSPI Flash and SRAM by the high-speed caches

- Up to 16 MBytes of external Flash are memory mapped into the CPU code space, supporting 8-bit, 16-bit and 32-bit access. Code execution is supported.

- Up to 8 MBytes of external Flash/SRAM are memory mapped into the CPU data space, supporting 8-bit, 16-bit and 32-bit access. Data read is supported on the Flash and SRAM. Data write is supported on the SRAM.

3.1.4 Memory Map

The structure of address mapping is shown in Figure 3. The memory and peripherals mapping of ESP32 is shown in Table 3.

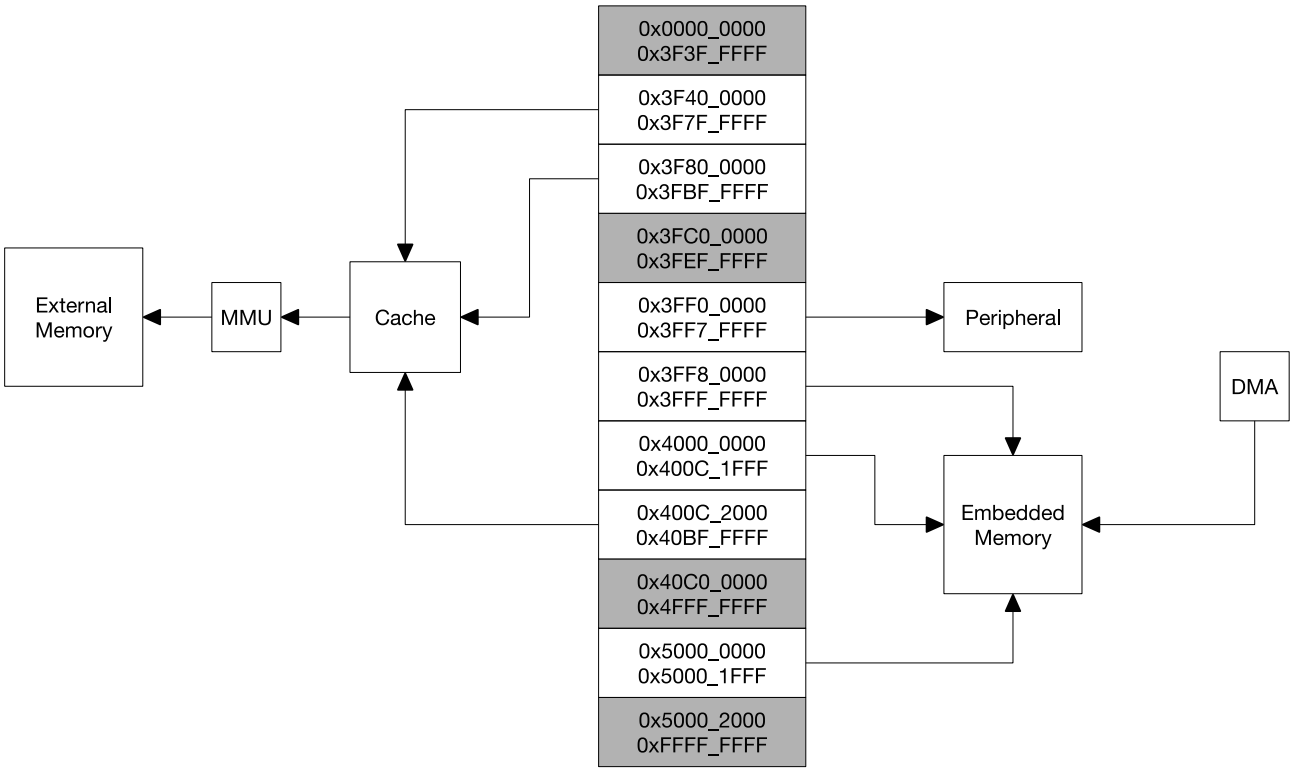


Figure 3: Address Mapping Structure

Table 3: Memory and Peripheral Mapping

Category	Target	Start Address	End Address	Size
Embedded Memory	Internal ROM 0	0x4000_0000	0x4005_FFFF	384 KB
	Internal ROM 1	0x3FF9_0000	0x3FF9_FFFF	64 KB
	Internal SRAM 0	0x4007_0000	0x4009_FFFF	192 KB
	Internal SRAM 1	0x3FFE_0000	0x3FFF_FFFF	128 KB
		0x400A_0000	0x400B_FFFF	
	Internal SRAM 2	0x3FFA_E000	0x3FFD_FFFF	200 KB
	RTC FAST Memory	0x3FF8_0000	0x3FF8_1FFF	8 KB
		0x400C_0000	0x400C_1FFF	
	RTC SLOW Memory	0x5000_0000	0x5000_1FFF	8 KB
External Memory	External Flash	0x3F40_0000	0x3F7F_FFFF	4 MB
		0x400C_2000	0x40BF_FFFF	11 MB
				248 KB
	External SRAM	0x3F80_0000	0x3FBF_FFFF	4 MB

Category	Target	Start Address	End Address	Size
Peripheral	DPort Register	0x3FF0_0000	0x3FF0_0FFF	4 KB
	AES Accelerator	0x3FF0_1000	0x3FF0_1FFF	4 KB
	RSA Accelerator	0x3FF0_2000	0x3FF0_2FFF	4 KB
	SHA Accelerator	0x3FF0_3000	0x3FF0_3FFF	4 KB
	Secure Boot	0x3FF0_4000	0x3FF0_4FFF	4 KB
	Cache MMU Table	0x3FF1_0000	0x3FF1_3FFF	16 KB
	PID Controller	0x3FF1_F000	0x3FF1_FFFF	4 KB
	UART0	0x3FF4_0000	0x3FF4_0FFF	4 KB
	SPI1	0x3FF4_2000	0x3FF4_2FFF	4 KB
	SPI0	0x3FF4_3000	0x3FF4_3FFF	4 KB
	GPIO	0x3FF4_4000	0x3FF4_4FFF	4 KB
	RTC	0x3FF4_8000	0x3FF4_8FFF	4 KB
	IO MUX	0x3FF4_9000	0x3FF4_9FFF	4 KB
	SDIO Slave	0x3FF4_B000	0x3FF4_BFFF	4 KB
	UDMA1	0x3FF4_C000	0x3FF4_CFFF	4 KB
	I2S0	0x3FF4_F000	0x3FF4_FFFF	4 KB
	UART1	0x3FF5_0000	0x3FF5_0FFF	4 KB
	I2C0	0x3FF5_3000	0x3FF5_3FFF	4 KB
	UDMA0	0x3FF5_4000	0x3FF5_4FFF	4 KB
	SDIO Slave	0x3FF5_5000	0x3FF5_5FFF	4 KB
	RMT	0x3FF5_6000	0x3FF5_6FFF	4 KB
	PCNT	0x3FF5_7000	0x3FF5_7FFF	4 KB
	SDIO Slave	0x3FF5_8000	0x3FF5_8FFF	4 KB
	LED PWM	0x3FF5_9000	0x3FF5_9FFF	4 KB
	Efuse Controller	0x3FF5_A000	0x3FF5_AFFF	4 KB
	Flash Encryption	0x3FF5_B000	0x3FF5_BFFF	4 KB
	PWM0	0x3FF5_E000	0x3FF5_EFFF	4 KB
	TIMG0	0x3FF5_F000	0x3FF5_FFFF	4 KB
	TIMG1	0x3FF6_0000	0x3FF6_0FFF	4 KB
	SPI2	0x3FF6_4000	0x3FF6_4FFF	4 KB
	SPI3	0x3FF6_5000	0x3FF6_5FFF	4 KB
	SYSCON	0x3FF6_6000	0x3FF6_6FFF	4 KB
	I2C1	0x3FF6_7000	0x3FF6_7FFF	4 KB
	SDMMC	0x3FF6_8000	0x3FF6_8FFF	4 KB
	EMAC	0x3FF6_9000	0x3FF6_AFFF	8 KB
	PWM1	0x3FF6_C000	0x3FF6_CFFF	4 KB
	I2S1	0x3FF6_D000	0x3FF6_DFFF	4 KB
	UART2	0x3FF6_E000	0x3FF6_EFFF	4 KB
	PWM2	0x3FF6_F000	0x3FF6_FFFF	4 KB
	PWM3	0x3FF7_0000	0x3FF7_0FFF	4 KB
	RNG	0x3FF7_5000	0x3FF7_5FFF	4 KB

3.2 Timers and Watchdogs

3.2.1 64-bit Timers

There are four general-purpose timers embedded in the ESP32. They are all 64-bit generic timers which are based on 16-bit prescalers and 64-bit auto-reload-capable up/downcounters.

The timers feature:

- A 16-bit clock prescaler, from 2 to 65536
- A 64-bit time-base counter
- Configurable up/down time-base counter: incrementing or decrmenting
- Halt and resume of time-base counter
- Auto-reload at alarming
- Software-controlled instant reload
- Level and edge interrupt generation

3.2.2 Watchdog Timers

The ESP32 has three watchdog timers: one in each of the two timer modules (called the Main Watchdog Timer, or MWDT) and one in the RTC module (called the RTC Watchdog Timer, or RWDT). These watchdog timers are intended to recover from an unforeseen fault, causing the application program to abandon its normal sequence. A watchdog timer has 4 stages. Each stage may take one of three or four actions on expiry of a programmed time period for this stage unless the watchdog is fed or disabled. The actions are: interrupt, CPU reset, and core reset, and system reset. Only the RWDT can trigger the system reset, and is able to reset the entire chip, including the RTC itself. A timeout value can be set for each stage individually.

During Flash boot the RWDT and the first MWDT start automatically in order to detect and recover from booting problems.

The ESP32 watchdogs have the following features:

- 4 stages, each can be configured or disabled separately
- Programmable time period for each stage
- One of 3 or 4 possible actions (interrupt, CPU reset, core reset, and system reset) on expiration of each stage
- 32-bit expiry counter
- Write protection, to prevent the RWDT and MWDT configuration from being inadvertently altered
- SPI Flash boot protection
  If the boot process from an SPI Flash does not complete within a predetermined time period, the watchdog will reboot the entire system.

3.3 System Clocks

3.3.1 CPU Clock

Upon reset, an external crystal clock source (2 MHz ~ 60 MHz), is selected as the default CPU clock. The external crystal clock source also connects to a PLL to generate a high frequency clock (typically 160 MHz).

In addition to this, ESP32 has an internal 8 MHz oscillator, of which the accuracy is guaranteed by design and is stable over temperature (within 1% accuracy). Hence, the application can then select from the external crystal clock source, the PLL clock or the internal 8 MHz oscillator. The selected clock source drives the CPU clock, directly or after division, depending on the application.

3.3.2 RTC Clock

The RTC clock has five possible sources:

- external low speed (32 kHz) crystal clock
- external crystal clock divided by 4
- internal RC oscillator (typically about 150 kHz and adjustable)
- internal 8 MHz oscillator
- internal 31.25 kHz clock (derived from the internal 8 MHz oscillator divided by 256)

When the chip is in the normal power mode and needs faster CPU accessing, the application can choose the external high speed crystal clock divided by 4 or the internal 8 MHz oscillator. When the chip operates in the low power mode, the application chooses the external low speed (32 kHz) crystal clock, the internal RC clock or the internal 31.25 kHz clock.

3.3.3 Audio PLL Clock

The audio clock is generated by the ultra low noise fractional-N PLL. The output frequency of the audio PLL is programmable, from 16 MHz to 128 MHz, given by the following formula:

$$f_{\text{out}} = \frac{f_{\text{xtal}} N_{\text{div}}}{M_{\text{div}} 2^{K_{\text{div}}}}$$

where f_{out} is the output frequency, f_{xtal} is the frequency of the crystal oscillator, and N_{div}, M_{div} and K_{div} are all integer values, configurable by registers.

3.4 Radio

The ESP32 radio consists of the following main blocks:

- 2.4 GHz receiver
- 2.4 GHz transmitter
- bias and regulators
- balun and transmit-receive switch
- clock generator

3.4.1 2.4 GHz Receiver

The 2.4 GHz receiver down-converts the 2.4 GHz RF signal to quadrature baseband signals and converts them to the digital domain with 2 high-resolution, high-speed ADCs. To adapt to varying signal channel conditions, RF filters, Automatic Gain Control (AGC), DC offset cancelation circuits and baseband filters are integrated within ESP32.

3.4.2 2.4 GHz Transmitter

The 2.4 GHz transmitter up-converts the quadrature baseband signals to the 2.4 GHz RF signal, and drives the antenna with a high powered Complementary Metal Oxide Semiconductor (CMOS) power amplifier. The use of digital calibration further improves the linearity of the power amplifier, enabling state-of-the-art performance of delivering +20.5 dBm of average power for 802.11b transmission and +17 dBm for 802.11n transmission.
Additional calibrations are integrated to cancel any imperfections of the radio, such as:

- Carrier leakage
- I/Q phase matching
- Baseband nonlinearities
- RF nonlinearities
- Antenna matching

These built-in calibration routines reduce the amount of time and required for product test and make test equipment unnecessary.

3.4.3 Clock Generator

The clock generator generates quadrature 2.4 GHz clock signals for the receiver and transmitter. All components of the clock generator are integrated on the chip, including all inductors, varactors, filters, regulators and dividers. The clock generator has built-in calibration and self test circuits. Quadrature clock phases and phase noise are optimized on-chip with patented calibration algorithms to ensure the best performance of the receiver and transmitter.

3.5 Wi-Fi

ESP32 implements TCP/IP, full 802.11 b/g/n/e/i WLAN MAC protocol, and Wi-Fi Direct specification. It supports Basic Service Set (BSS) STA and SoftAP operations under the Distributed Control Function (DCF) and P2P group operation compliant with the latest Wi-Fi P2P protocol.

Passive or active scanning, as well as the P2P discovery procedure are performed autonomously when initiated by appropriate commands. Power management is handled with minimum host interaction to minimize active duty period.

3.5.1 Wi-Fi Radio and Baseband

The ESP32 Wi-Fi Radio and Baseband support the following features:

- 802.11b and 802.11g data-rates
- 802.11n MCS0-7 in both 20 MHz and 40 MHz bandwidth
- 802.11n MCS32
- 802.11n 0.4 μS guard-interval
- Data-rate up to 150 Mbps
- Receiving STBC 2x1
- Up to 21 dBm transmitting power
- Adjustable transmitting power

- Antenna diversity and selection (software-managed hardware)

3.5.2 Wi-Fi MAC

The ESP32 Wi-Fi MAC applies low level protocol functions automatically as follows:

- Request To Send (RTS), Clear To Send (CTS) and Acknowledgement (ACK/BA)
- Fragmentation and defragmentation
- Aggregation AMPDU and AMSDU
- WMM, U-APSD
- 802.11 e: QoS for wireless multimedia technology
- CCMP (CBC-MAC, counter mode), TKIP (MIC, RC4), WAPI (SMS4), WEP (RC4) and CRC
- Frame encapsulation (802.11h/RFC 1042)
- Automatic beacon monitoring/scanning

3.5.3 Wi-Fi Firmware

The ESP32 Wi-Fi Firmware provides the following functions:

- Infrastructure BSS Station mode / P2P mode / softAP mode support
- P2P Discovery, P2P Group Owner, P2P Group Client and P2P Power Management
- WPA/WPA2-Enterprise and WPS driver
- Additional 802.11i security features such as pre-authentication and TSN
- Open interface for various upper layer authentication schemes over EAP such as TLS, PEAP, LEAP, SIM, AKA or customer specific
- Clock/power gating combined with 802.11-compliant power management dynamically adapted to current connection condition providing minimal power consumption
- Adaptive rate fallback algorithm sets the optimal transmission rate and transmit power based on actual Signal Noise Ratio (SNR) and packet loss information
- Automatic retransmission and response on MAC to avoid packet discarding on slow host environment

3.5.4 Packet Traffic Arbitration (PTA)

ESP32 has a configurable Packet Traffic Arbitration (PTA) that provides flexible and exact timing Bluetooth co-existence support. It is a combination of both Frequency Division Multiplexing (FDM) and Time Division Multiplexing (TDM), and coordinates the protocol stacks.

- It is preferable that Wi-Fi works in the 20 MHz bandwidth mode to decrease its interference with BT.
- BT applies AFH (Adaptive Frequency Hopping) to avoid using the channels within Wi-Fi bandwidth.
- Wi-Fi MAC limits the time duration of Wi-Fi packets, and does not transmit the long Wi-Fi packets by the lowest data-rates.
- Normally BT packets are of higher priority than normal Wi-Fi packets.
- Protect the critical Wi-Fi packets, including beacon transmission and receiving, ACK/BA transmission and receiving.

- Protect the highest BT packets, including inquiry response, page response, LMP data and response, park beacons, the last poll period, SCO/eSCO slots, and BLE event sequence.
- Wi-Fi MAC applies CTS-to-self packet to protect the time duration of BT transfer.
- In the P2P Group Own (GO) mode, Wi-Fi MAC applies a Notice of Absence (NoA) packet to disable Wi-Fi transfer to reserve time for BT.
- In the STA mode, Wi-Fi MAC applies a NULL packet with the Power-Save bit to disable WiFi transfer to reserve time for BT.

3.6 Bluetooth

ESP32 integrates Bluetooth link controller and Bluetooth baseband, which carry out the baseband protocols and other low-level link routines, such as modulation/demodulation, packets processing, bit stream processing, frequency hopping, etc.

3.6.1 Bluetooth Radio and Baseband

The ESP32 Bluetooth Radio and Baseband support the following features:

- Class-1, class-2 and class-3 transmit output powers and over 30 dB dynamic control range
- $\pi/4$ DQPSK and 8 DPSK modulation
- High performance in NZIF receiver sensitivity with over 98 dB dynamic range
- Class-1 operation without external PA
- Internal SRAM allows full speed data transfer, mixed voice and data, and full piconet operation
- Logic for forward error correction, header error control, access code correlation, CRC, demodulation, encryption bit stream generation, whitening and transmit pulse shaping
- ACL, SCO, eSCO and AFH
- A-law, μ-law and CVSD digital audio CODEC in PCM interface
- SBC audio CODEC
- Power management for low power applications
- SMP with 128-bit AES

3.6.2 Bluetooth Interface

- Provides UART HCI interface, up to 4 Mbps
- Provides SDIO / SPI HCI interface
- Provides I2C interface for the host to do configuration
- Provides PCM / I2S audio interface

3.6.3 Bluetooth Stack

The Bluetooth stack of ESP32 is compliant with Bluetooth v4.2 BR / EDR and BLE specification.

3.6.4 Bluetooth Link Controller

The link controller operates in three major states: standby, connection and sniff. It enables multi connection and other operations like inquiry, page, and secure simple pairing, and therefore enables Piconet and Scatternet. Below are the features:

- Classic Bluetooth
  - Device Discovery (inquiry and inquiry scan)
  - Connection establishment (page and page scan)
  - Multi connections
  - Asynchronous data reception and transmission
  - Synchronous links (SCO/eSCO)
  - Master/Slave Switch
  - Adaptive Frequency Hopping and Channel assessment
  - Broadcast encryption
  - Authentication and encryption
  - Secure Simple Pairing
  - Multi-point and scatternet management
  - Sniff mode
  - Connectionless Slave Broadcast (transmitter and receiver)
  - Enhanced power control
  - Ping
- Bluetooth Low Energy
  - Advertising
  - Scanning
  - Multiple connections
  - Asynchronous data reception and transmission
  - Adaptive Frequency Hopping and Channel assessment
  - Connection parameter update
  - Date Length Extension
  - Link Layer Encryption
  - LE Ping

3.7 RTC and Low-Power Management

With the advanced power management technologies, ESP32 can switch between different power modes (see Table 4).

- Power mode
  - Active mode: The chip radio is powered on. The chip can receive, transmit, or listen.
  - Modem-sleep mode: The CPU is operational and the clock is configurable. The Wi-Fi/Bluetooth baseband and radio are disabled.
  - Light-sleep mode: The CPU is paused. The RTC and ULP-coprocessor are running. Any wake-up events (MAC, host, RTC timer, or external interrupts) will wake up the chip.
  - Deep-sleep mode: Only RTC is powered on. Wi-Fi and Bluetooth connection data are stored in RTC memory. The ULP-coprocessor can work.
  - Hibernation mode: The internal 8MHz oscillator and ULP-coprocessor are disabled. The RTC recovery memory are power-down. Only one RTC timer on the slow clock and some RTC GPIOs are active. The RTC timer or the RTC GPIOs can wake up the chip from the Hibernation mode.
- Sleep Pattern
  - Association sleep pattern: The power mode switches between the active mode and Modem-sleep/Light-sleep mode during this sleep pattern. The CPU, Wi-Fi, Bluetooth, and radio are woken up at predetermined intervals to keep Wi-Fi/BT connections alive.
  - ULP sensor-monitored pattern: The main CPU is in the Deep-sleep mode. The ULP co-processor does sensor measurements and wakes up the main system, based on the measured data from sensors.

Table 4: Functionalities Depending on the Power Modes

Power mode	Active	Modem-sleep	Light-sleep	Deep-sleep	Hibernation
Sleep pattern	Association sleep pattern			ULP sensor-monitored pattern	-
CPU	ON	PAUSE	ON	OFF	OFF
Wi-Fi/BT baseband and radio	ON	OFF	OFF	OFF	OFF
RTC	ON	ON	ON	ON	OFF
ULP co-processor	ON	ON	ON	ON/OFF	OFF

The power consumption varies with different power modes/sleep patterns and work status of functional modules (see Table 5).

Table 5: Power Consumption by Power Modes

Power mode	Description	Power consumption
Active (RF working)	Wi-Fi Tx packet 13 dBm ~ 21 dBm	160 ~ 260 mA
	Wi-Fi / BT Tx packet 0 dBm	120 mA
	Wi-Fi / BT Rx and listening	80 ~ 90 mA
	Association sleep pattern (by Light-sleep)	0.9 mA@DTIM3, 1.2 mA@DTIM1
Modem-sleep	The CPU is powered on.	Max speed: 20 mA
		Normal speed: 5 ~ 10 mA
		Slow speed: 3 mA
Light-sleep	-	0.8 mA
Deep-sleep	The ULP co-processor is powered on.	0.15 mA
	ULP sensor-monitored pattern	25 μA @1% duty
	RTC timer + RTC memory	10 μA
Hibernation	RTC timer only	2.5 μA

Note:
For more information about RF power consumption, refer to Section 5.3 RF Power Consumption Specifications.

4. Peripheral Interface

4.1 General Purpose Input / Output Interface (GPIO)

ESP32 has 48 GPIO pins which can be assigned to various functions by programming the appropriate registers. There are several kinds of GPIOs: digital only GPIOs, analog enabled GPIOs, capacitive touch enabled GPIOs, etc. Analog enabled GPIOs can be configured as digital GPIOs. Capacitive touch enabled GPIOs can be configured as digital GPIOs.

Each digital enabled GPIO can be configured to internal pull-up or pull-down, or set to high impedance. When configured as an input, the input value can be read through the register. The input can also be set to edge-trigger or level-trigger to generate CPU interrupts. In short, the digital IO pins are bi-directional, non-inverting and tristate, including input and output buffer with tristate control. These pins can be multiplexed with other functions, such as the SDIO interface, UART, SI, etc. For low power operations, the GPIOs can be set to hold their states.

4.2 Analog-to-Digital Converter (ADC)

ESP32 integrates 12-bit SAR ADCs and supports measurements on 18 channels (analog enabled pins). Some of these pins can be used to build a programmable gain amplifier which is used for the measurement of small analog signals. The ULP-coprocessor in ESP32 is also designed to measure the voltages while operating in the sleep mode, to enable low power consumption; the CPU can be woken up by a threshold setting and/or via other triggers.

With the appropriate setting, the ADCs and the amplifier can be configured to measure voltages for a maximum of 18 pins.

4.3 Ultra Low Noise Analog Pre-Amplifier

ESP32 integrates an ultra low noise analog pre-amplifier that outputs to the ADC. The amplification ratio is given by the size of a pair of sampling capacitors that are placed off-chip. By using a larger capacitor, the sampling noise is reduced, but the settling time will be increased. The amplification ratio is also limited by the amplifier which peaks at about 60 dB gain.

4.4 Hall Sensor

ESP32 integrates a Hall sensor based on an N-carrier resistor. When the chip is in the magnetic field, the Hall sensor develops a small voltage laterally on the resistor, which can be directly measured by the ADC, or amplified by the ultra low noise analog pre-amplifier and then measured by the ADC.

4.5 Digital-to-Analog Converter (DAC)

Two 8-bit DAC channels can be used to convert two digital signals into two analog voltage signal outputs. The design structure is composed of integrated resistor strings and a buffer. This dual DAC supports power supply as input voltage reference and can drive other circuits. The dual channels support independent conversions.

4.6 Temperature Sensor

The temperature sensor generates a voltage that varies with temperature. The voltage is internally converted via an analog-to-digital converter into a digital code.

The temperature sensor has a range of -40°C to 125°C. As the offset of the temperature sensor varies from chip to chip due to process variation, together with the heat generated by the Wi-Fi circuitry itself (which affects measurements), the internal temperature sensor is only suitable for applications that detect temperature changes instead of absolute temperatures and for calibration purposes as well.

However, if the user calibrates the temperature sensor and uses the device in a minimally powered-on application, the results could be accurate enough.

4.7 Touch Sensor

ESP32 offers 10 capacitive sensing GPIOs which detect capacitive variations introduced by the GPIO's direct contact or close proximity with a finger or other objects. The low noise nature of the design and high sensitivity of the circuit allow relatively small pads to be used. Arrays of pads can also be used so that a larger area or more points can be detected. The 10 capacitive sensing GPIOs are listed in Table 6.

Table 6: Capacitive Sensing GPIOs Available on ESP32

Capacitive sensing signal name	Pin name
T0	GPIO4
T1	GPIO0
T2	GPIO2
T3	MTDO
T4	MTCK
T5	MTD1
T6	MTMS
T7	GPIO27
T8	32K_XN
T9	32K_XP

Note:
For more information about the touch sensor design and layout, refer to Appendix A Touch Sensor.

4.8 Ultra-Lower-Power Coprocessor

The ULP processor and RTC memory remains powered on during the Deep-sleep mode. Hence, the developer can store a program for the ULP processor in the RTC memory to access the peripheral devices, internal timers and internal sensors during the Deep-sleep mode. This is useful for designing applications where the CPU needs to be woken up by an external event, or timer, or a combination of these events, while maintaining minimal power consumption.

4.9 Ethernet MAC Interface

An IEEE-802.3-2008-compliant Media Access Controller (MAC) is provided for Ethernet LAN communications. ESP32 requires an external physical interface device (PHY) to connect to the physical LAN bus (twisted-pair, fiber, etc.). The PHY is connected to ESP32 through 17 signals of MII or 9 signals of RMII. With the Ethernet MAC (EMAC) interface, the following features are supported:

- 10 Mbps and 100 Mbps rates
- Dedicated DMA controller allowing high-speed transfer between the dedicated SRAM and Ethernet MAC
- Tagged MAC frame (VLAN support)
- Half-duplex (CSMA/CD) and full-duplex operation
- MAC control sublayer (control frames)
- 32-bit CRC generation and removal
- Several address filtering modes for physical and multicast address (multicast and group addresses)
- 32-bit status code for each transmitted or received frame
- Internal FIFOs to buffer transmit and receive frames. The transmit FIFO and the receive FIFO are both 512 words (32-bit)
- Hardware PTP (precision time protocol) in accordance with IEEE 1588 2008 (PTP V2)
- 25 MHz/50 MHz clock output

4.10 SD/SDIO/MMC Host Controller

An SD/SDIO/MMC host controller is available on ESP32 which supports the following features:

- Secure Digital memory (SD mem Version 3.0 and Version 3.01)
- Secure Digital I/O (SDIO Version 3.0)
- Consumer Electronics Advanced Transport Architecture (CE-ATA Version 1.1)
- Multimedia Cards (MMC Version 4.41, eMMC Version 4.5 and Version 4.51)

The controller allows clock output at up to 80 MHz and in three different data-bus modes: 1-bit, 4-bit and 8-bit. It supports two SD/SDIO/MMC4.41 cards in 4-bit data-bus mode. It also supports one SD card operating at 1.8 V level.

4.11 Universal Asynchronous Receiver Transmitter (UART)

ESP32 has three UART interfaces, i.e. UART0, UART1 and UART2, which provide asynchronous communication (RS232 and RS485) and IrDA support, and communicate at up to 5 Mbps. UART provides hardware management of the CTS and RTS signals and software flow control (XON and XOFF). All of the interfaces can be accessed by the DMA controller or directly by CPU.

4.12 I2C Interface

ESP32 has two I2C bus interfaces which can serve as I2C master or slave depending on the user's configuration. The I2C interfaces support:

- Standard mode (100 kbit/s)
- Fast mode (400 kbit/s)
- Up to 5 MHz, but constrained by SDA pull up strength
- 7-bit/10-bit addressing mode
- Dual addressing mode

Users can program command registers to control I2C interfaces to have more flexibility.

4.13 I2S Interface

Two standard I2S interfaces are available in ESP32. They can be operated in the master or slave mode, in full duplex and half-duplex communication modes, and can be configured to operate with an 8-/16-/32-/40-/48-bit resolution as input or output channels. BCK clock frequency from 10 kHz up to 40 MHz are supported. When one or both of the I2S interfaces are configured in the master mode, the master clock can be output to the external DAC/CODEC.

Both of the I2S interfaces have dedicated DMA controllers. PDM and BT PCM interfaces are supported.

4.14 Infrared Remote Controller

The infrared remote controller supports eight channels of infrared remote transmission and receiving. Through programming the pulse waveform, it supports various infrared protocols. Eight channels share a 512 x 32-bit block of memory to store the transmitting or receiving waveform.

4.15 Pulse Counter

The pulse counter captures pulse and counts pulse edges through seven modes. It has 8 channels; each channel captures four signals at a time. The four input signals include two pulse signals and two control signals. When the counter reaches a defined threshold, an interrupt is generated.

4.16 Pulse Width Modulation (PWM)

The Pulse Width Modulation (PWM) controller can be used for driving digital motors and smart lights. The controller consists of PWM timers, the PWM operator and a dedicated capture sub-module. Each timer provides timing in synchronus or independent form, and each PWM operator generates the waveform for one PWM channel. The dedicated capture sub-module can accurately capture external timing events.

4.17 LED PWM

The LED PWM controller can generate 16 independent channels of digital waveforms with the configurable periods and configurable duties.

The 16 channels of digital waveforms operate at 80 MHz APB clock, among which 8 channels have the option of using the 8 MHz oscillator clock. Each channel can select a 20-bit timer with configurable counting range and its accuracy of duty can be up to 16 bits with the 1 ms period.

The software can change the duty immediately. Moreover, each channel supports step-by-step duty increasing or decreasing automatically. It is useful for the LED RGB color gradient generator.

4.18 Serial Peripheral Interface (SPI)

ESP32 features three SPIs (SPI, HSPI and VSPI) in slave and master modes in 1-line full-duplex and 1/2/4-line half-duplex communication modes. These SPIs also support the following general-purpose SPI features:

- 4 timing modes of the SPI format transfer that depend on the polarity (POL) and the phase (PHA)
- up to 80 MHz and the divided clocks of 80 MHz
- up to 64-Byte FIFO

All SPIs can also be used to connect to the external Flash/SRAM and LCD. Each SPI can be served by DMA controllers.

4.19 Accelerator

ESP32 is equipped with hardware accelerators of general algorithms, such as AES (FIPS PUB 197), SHA (FIPS PUB 180-4), RSA, and ECC, which support independent arithmetic such as Big Integer Multiplication and Big Integer Modular Multiplication. The maximum operation length for RSA, ECC, Big Integer Multiply and Big Integer Modular Multiplication is 4096 bits.

The hardware accelerators greatly improve operation speed and reduce software complexity. They also support code encryption and dynamic decryption which ensures that codes in the Flash will not be stolen.

5. Electrical Characteristics

Note:
The specifications in this charpter are tested in general condition: V_{BAT} = 3.3V, T_A = 27°C, unless otherwise specified.

5.1 Absolute Maximum Ratings

Table 7: Absolute Maximum Ratings

Parameter	Symbol	Min	Max	Unit
Input low voltage	V_{IL}	-0.3	0.25×V_{IO}	V
Input high voltage	V_{IH}	0.75×V_{IO}	3.3	V
Input leakage current	I_{IL}	-	50	nA
Output low voltage	V_{OL}	-	0.1×V_{IO}	V
Output high voltage	V_{OH}	0.8×V_{IO}	-	V
Input pin capacitance	C_{pad}	-	2	pF
VDDIO	V_{IO}	1.8	3.3	V
Maximum drive capability	I_{MAX}	-	12	mA
Storage temperature range	T_{STR}	-40	150	°C

5.2 Recommended Operating Conditions

Table 8: Recommended Operating Conditions

Parameter	Symbol	Min	Typ	Max	Unit
Battery regulator supply voltage	V_{BAT}	2.8	3.3	3.6	V
I/O supply voltage	V_{IO}	1.8	3.3	3.6	V
Operating temperature range	T_{OPR}	-40	-	125	°C
CMOS low level input voltage	V_{IL}	0	-	0.3 x V_{IO}	V
CMOS high level input voltage	V_{IH}	0.7 x V_{IO}	-	V_{IO}	V
CMOS threshold voltage	V_{TH}	-	0.5 x V_{IO}	-	V

5.3 RF Power Consumption Specifications

The current consumption measurements are conducted with 3.0 V supply and 25°C ambient, at antenna port. All the transmitters' measurements are based on 90% duty cycle and continuous transmit mode.

Table 9: RF Power Consumption Specifications

Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	225	-	mA
Transmit 802.11b, CCK 11 Mbps, POUT = +18.5 dBm	-	205	-	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	-	160	-	mA
Transmit 802.11n, MCS7, POUT = +14 dBm	-	152	-	mA
Receive 802.11b, packet length = 1024 bytes, -80 dBm	-	85	-	mA
Receive 802.11g, packet length = 1024 bytes, -70 dBm	-	85	-	mA
Receive 802.11n, packet length = 1024 bytes, -65 dBm	-	80	-	mA
Receive 802.11n HT40, packet length = 1024 bytes, -65 dBm	-	80	-	mA

5.4 Wi-Fi Radio

Table 10: Wi-Fi Radio Characteristics

Description	Min	Typical	Max	Unit
Input frequency	2412	-	2484	MHz
Input impedance	-	50	-	Ω
Input reflection	-	-	-10	dB
Output power of PA for 72.2 Mbps	15.5	16.5	17.5	dBm
Output power of PA for 11b mode	19.5	20.5	21.5	dBm
DSSS, 1 Mbps	-	-98	-	dBm
CCK, 11 Mbps	-	-91	-	dBm
OFDM, 6 Mbps	-	-93	-	dBm
OFDM, 54 Mbps	-	-75	-	dBm
HT20, MCS0	-	-93	-	dBm
HT20, MCS7	-	-73	-	dBm
HT40, MCS0	-	-90	-	dBm
HT40, MCS7	-	-70	-	dBm
MCS32	-	-89	-	dBm
OFDM, 6 Mbps	-	37	-	dB
OFDM, 54 Mbps	-	21	-	dB
HT20, MCS0	-	37	-	dB
HT20, MCS7	-	20	-	dB

5.5 Bluetooth Radio

5.5.1 Receiver - Basic Data Rate

Table 11: Receiver Characteristics-Basic Data Rate

Parameter	Conditions	Min	Typ	Max	Unit
Sensitivity @0.1% BER	-	-	-98	-	dBm
Maximum received signal @0.1% BER	-	0	-	-	dBm
Co-channel C/I	-	-	+7	-	dB
Adjacent channel selectivity C/I	F = F0 + 1 MHz	-	-	-6	dB
	F = F0 - 1 MHz	-	-	-6	dB
	F = F0 + 2 MHz	-	-	-25	dB
	F = F0 - 2 MHz	-	-	-33	dB
	F = F0 + 3 MHz	-	-	-25	dB
	F = F0 - 3 MHz	-	-	-45	dB
Out-of-band blocking performance	30 MHz ~ 2000 MHz	-10	-	-	dBm
	2000 MHz ~ 2400 MHz	-27	-	-	dBm
	2500 MHz ~ 3000 MHz	-27	-	-	dBm
	3000 MHz ~ 12.5 GHz	-10	-	-	dBm
Intermodulation	-	-36	-	-	dBm

5.5.2 Transmitter - Basic Data Rate

Table 12: Transmitter Characteristics - Basic Data Rate

Parameter	Conditions	Min	Typ	Max	Unit
RF transmit power	-	-	+4	+4	dBm
RF power control range	-	-	25	-	dB
20 dB bandwidth	-	-	0.9	-	MHz
Adjacent channel transmit power	F = F0 + 1 MHz	-	-24	-	dBm
	F = F0 - 1 MHz	-	-16.1	-	dBm
	F = F0 + 2 MHz	-	-40.8	-	dBm
	F = F0 - 2 MHz	-	-35.6	-	dBm
	F = F0 + 3 MHz	-	-45.7	-	dBm
	F = F0 - 3 MHz	-	-40.2	-	dBm
	F = F0 + > 3 MHz	-	-45.6	-	dBm
	F = F0 - > 3 MHz	-	-44.6	-	dBm
Δ f1$_{avg}$	-	-	-	155	kHz
Δ f2$_{max}$	-	133.7	-	-	kHz
Δ f2$_{avg}$/Δ f1$_{avg}$	-	-	0.92	-	-
ICFT	-	-	-7	-	kHz
Drift rate	-	-	0.7	-	kHz/50 μs
Drift (1 slot packet)	-	-	6	-	kHz
Drift (5 slot packet)	-	-	6	-	kHz

5.5.3 Receiver - Enhanced Data Rate

Table 13: Receiver Characteristics - Enhanced Data Rate

Parameter	Conditions	Min	Typ	Max	Unit
π/4 DQPSK					
Sensitivity @0.01% BER	-	-	-98	-	dBm
Maximum received signal @0.1% BER	-	-	0	-	dBm
Co-channel C/I	-	-	11	-	dB
Adjacent channel selectivity C/I	F = F0 + 1 MHz	-	-7	-	dB
	F = F0 - 1 MHz	-	-7	-	dB
	F = F0 + 2 MHz	-	-25	-	dB
	F = F0 - 2 MHz	-	-35	-	dB
	F = F0 + 3 MHz	-	-25	-	dB
	F = F0 - 3 MHz	-	-45	-	dB
8DPSK					
Sensitivity @0.01% BER	-	-	-84	-	dBm
Maximum received signal @0.1% BER	-	0	-	-	dBm
C/I c-channel	-	-	18	-	dB
Adjacent channel selectivity C/I	F = F0 + 1 MHz	-	2	-	dB
	F = F0 - 1 MHz	-	2	-	dB
	F = F0 + 2 MHz	-	-25	-	dB
	F = F0 - 2 MHz	-	-25	-	dB
	F = F0 + 3 MHz	-	-25	-	dB
	F = F0 - 3 MHz	-	-38	-	dB

5.5.4 Transmitter - Enhanced Data Rate

Table 14: Transmitter Characteristics - Enhanced Data Rate

Parameter	Conditions	Min	Typ	Max	Unit
Maximum RF transmit power	-	-	+2	-	dBm
Relative transmit control	-	-	-1.5	-	dB
π/4 DQPSK max w0	-	-	-0.72	-	kHz
π/4 DQPSK max wi	-	-	-6	-	kHz
π/4 DQPSK max |wi + w0|	-	-	-7.42	-	kHz
8DPSK max w0	-	-	0.7	-	kHz
8DPSK max wi	-	-	-9.6	-	kHz
8DPSK max |wi + w0|	-	-	-10	-	kHz
π/4 DQPSK modulation accuracy	RMS DEVM	-	4.28	-	%
	99% DEVM	-	-	30	%
	Peak DEVM	-	13.3	-	%
8 DPSK modulation accuracy	RMS DEVM	-	5.8	-	%
	99% DEVM	-	-	20	%
	Peak DEVM	-	14	-	%

Parameter	Conditions	Min	Typ	Max	Unit
In-band spurious emissions	F = F0 + 1 MHz	-	-34	-	dBm
	F = F0 - 1 MHz	-	-40.2	-	dBm
	F = F0 + 2 MHz	-	-34	-	dBm
	F = F0 - 2 MHz	-	-36	-	dBm
	F = F0 + 3 MHz	-	-38	-	dBm
	F = F0 - 3 MHz	-	-40.3	-	dBm
	F = F0 +/- > 3 MHz	-	-	-41.5	dBm
EDR differential phase coding	-	-	100	-	%

5.6 Bluetooth LE Radio

5.6.1 Receiver

Table 15: Receiver Characteristics - BLE

Parameter	Conditions	Min	Typ	Max	Unit
Sensitivity @0.1% BER	-	-	-98	-	dBm
Maximum received signal @0.1% BER	-	0	-	-	dBm
Co-channel C/I	-	-	+10	-	dB
Adjacent channel selectivity C/I	F = F0 + 1 MHz	-	-5	-	dB
	F = F0 - 1 MHz	-	-5	-	dB
	F = F0 + 2 MHz	-	-25	-	dB
	F = F0 - 2 MHz	-	-35	-	dB
	F = F0 + 3 MHz	-	-25	-	dB
	F = F0 - 3 MHz	-	-45	-	dB
Out-of-band blocking performance	30 MHz ~ 2000 MHz	-10	-	-	dBm
	2000 MHz ~ 2400 MHz	-27	-	-	dBm
	2500 MHz ~ 3000 MHz	-27	-	-	dBm
	3000 MHz ~ 12.5 GHz	-10	-	-	dBm
Intermodulation	-	-36	-	-	dBm

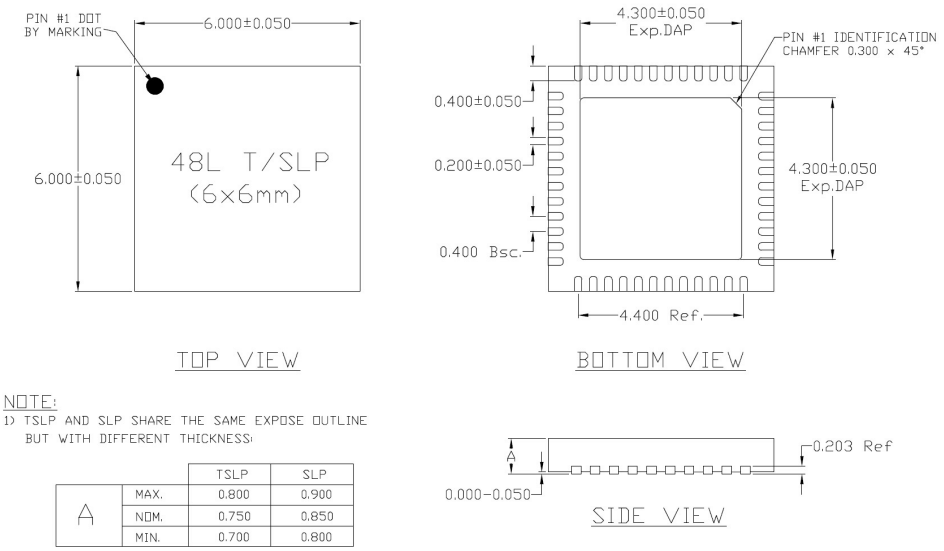
5.6.2 Transmitter

Table 16: Transmitter Characteristics - BLE

Parameter	Conditions	Min	Typ	Max	Unit
RF transmit power	-	-	+7.5	+10	dBm
RF power control range	-	-	25	-	dB
Adjacent channel transmit power	F = F0 + 1 MHz	-	-14.6	-	dBm
	F = F0 - 1 MHz	-	-12.7	-	dBm
	F = F0 + 2 MHz	-	-44.3	-	dBm
	F = F0 - 2 MHz	-	-38.7	-	dBm
	F = F0 + 3 MHz	-	-49.2	-	dBm
	F = F0 - 3 MHz	-	-44.7	-	dBm
	F = F0 + > 3 MHz	-	-50	-	dBm

Parameter	Conditions	Min	Typ	Max	Unit
	F = F0 - > 3 MHz	-	-50	-	dBm
Δ f1$_{avg}$	-	-	-	265	kHz
Δ f2$_{max}$	-	247	-	-	kHz
Δ f2$_{avg}$/Δ f1$_{avg}$	-	-	-0.92	-	-
ICFT	-	-	-10	-	kHz
Drift rate	-	-	0.7	-	kHz/50 μs
Drift	-	-	2	-	kHz

6. Package Information



		TSLP	SLP
A	MAX.	0.800	0.900
	NOM.	0.750	0.850
	MIN.	0.700	0.800

Figure 4: QFN48 (6x6 mm) Package

7. Supported Resources

7.1 Related Documentation

The following link provides related documents of ESP32.

- ESP32 Documentation
  All the available documentation and other resources of ESP32

7.2 Community Resources

The following links connect to ESP32 community resources.

- ESP32 Online Community
  An Engineer-to-Engineer (E2E) Community for ESP32 where you can ask questions, share knowledge, explore ideas and help solve problems with fellow engineers.
- ESP32 Github
  ESP32 development projects are freely distributed under Espressif's MIT license on Github. It is established to help developers get started with ESP32 and foster innovation and the growth of general knowledge about the hardware and software surrounding these devices.

Appendix A - Touch Sensor

A touch sensor system is built on a substrate which carries electrodes and relevant connections with a flat protective surface. When a user touches the surface, the capacitance variation is triggered, and a binary signal is generated to indicate whether the touch is valid.

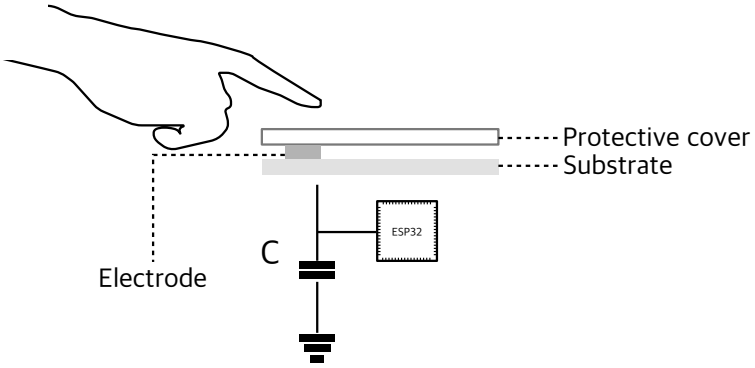


Figure 5: A Typical Touch Sensor Application

In order to prevent capacitive coupling and other electrical interference to the sensitivity of the touch sensor system, the following factors should be taken into account.

A.1. Electrode Pattern

The proper size and shape of an electrode helps improve system sensitivity. Round, oval, or shapes similar to a human fingertip is commonly applied. Large size or irregular shape might lead to incorrect responses from nearby electrodes.

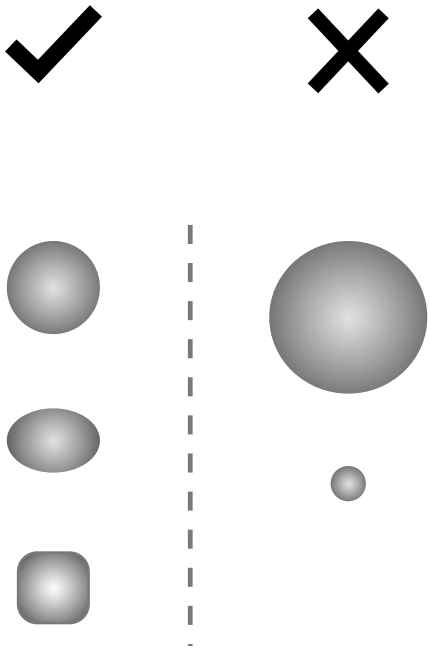


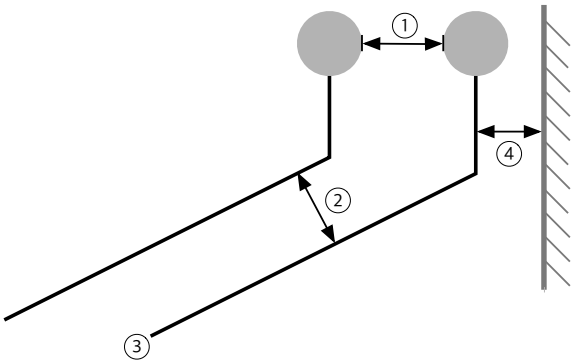
Figure 6: Electrode Pattern Requirements

Note:
The examples illustrated in Figure 6 are not of actual scale. It is suggested that users take a human fingertip as reference.

A.2. PCB Layout

The recommendations for correctly routing sensing tracks of electrodes are as follows:

- Close proximity between electrodes may lead to crosstalk between electrodes and false touch detections. The distance between electrodes should be at least twice the thickness of the panel used.
- The width of a sensor track creates parasitic capacitance, which could vary with manufacturing processes. The thinner the track is, the less capacitive coupling it generates. The track width should be kept as thin as possible and the length should not exceed 10cm to accommodate.
- We should avoid coupling between lines of high frequency signals. The sensing tracks should be routed parallel to each other on the same layer and the distance between the tracks should be at least twice the width of the track.
- When designing a touch sensor device, there should be no components adjacent to or underneath the electrodes.
- Do not ground the touch sensor device. It is preferable that no ground layer be placed under the device, unless there is a need to isolate it. Parasitic capacitance generated between the touch sensor device and the ground degrades sensitivity.



① Distance between electrodes – Twice the thickness of the panel
② Distance between tracks – Twice the track width
③ Width of the track (electrode wiring) – As thin as possible
④ Distance between track and ground plane – 2mm at a minimum

Figure 7: Sensor Track Routing Requirements

Appendix B - Code Examples

B.1. Input

```
>python esptool.py -p dev/tty8 -b 115200 write_Flash -c ESP32 -ff 40m -fm qio -fs 2MB
0x0 ~/Workspace/ESP32_BIN/boot.bin
0x04000 ~/Workspace/ESP32_BIN/drom0.bin
0x40000 ~/Workspace/ESP32_BIN/bin/irom0_Flash.bin
0xFC000 ~/Workspace/ESP32_BIN/blank.bin
0x1FC000 ~/Workspace/ESP32_BIN/esp_init_data_default.bin
```

B.2. Output

```
Connecting...
Erasing Flash...
Wrote 3072 bytes at 0x00000000 in 0.3 seconds (73.8 kbit/s)...
Erasing Flash...
Wrote 395264 bytes at 0x04000000 in 43.2 seconds (73.2 kbit/s)...
Erasing Flash...
Wrote 1024 bytes at 0x40000000 in 0.1 seconds (74.5 kbit/s)...
Erasing Flash...
Wrote 4096 bytes at 0xfc000000 in 0.4 seconds (73.5 kbit/s)...
Erasing Flash...
Wrote 4096 bytes at 0x1fc00000 in 0.5 seconds (73.8 kbit/s)...
Leaving...
```

PCA9685

16-channel, 12-bit PWM Fm+ I²C-bus LED controller

Rev. 4 — 16 April 2015 Product data sheet

1. General description

The PCA9685 is an I²C-bus controlled 16-channel LED controller optimized for Red/Green/Blue/Amber (RGBA) color backlighting applications. Each LED output has its own 12-bit resolution (4096 steps) fixed frequency individual PWM controller that operates at a programmable frequency from a typical of 24 Hz to 1526 Hz with a duty cycle that is adjustable from 0 % to 100 % to allow the LED to be set to a specific brightness value. All outputs are set to the same PWM frequency.

Each LED output can be off or on (no PWM control), or set at its individual PWM controller value. The LED output driver is programmed to be either open-drain with a 25 mA current sink capability at 5 V or totem pole with a 25 mA sink, 10 mA source capability at 5 V. The PCA9685 operates with a supply voltage range of 2.3 V to 5.5 V and the inputs and outputs are 5.5 V tolerant. LEDs can be directly connected to the LED output (up to 25 mA, 5.5 V) or controlled with external drivers and a minimum amount of discrete components for larger current or higher voltage LEDs.

The PCA9685 is in the new Fast-mode Plus (Fm+) family. Fm+ devices offer higher frequency (up to 1 MHz) and more densely populated bus operation (up to 4000 pF).

Although the PCA9635 and PCA9685 have many similar features, the PCA9685 has some unique features that make it more suitable for applications such as LCD or LED backlighting and Ambilight:

- The PCA9685 allows staggered LED output on and off times to minimize current surges. The on and off time delay is independently programmable for each of the 16 channels. This feature is not available in PCA9635.
- The PCA9685 has 4096 steps (12-bit PWM) of individual LED brightness control. The PCA9635 has only 256 steps (8-bit PWM).
- When multiple LED controllers are incorporated in a system, the PWM pulse widths between multiple devices may differ if PCA9635s are used. The PCA9685 has a programmable prescaler to adjust the PWM pulse widths of multiple devices.
- The PCA9685 has an external clock input pin that will accept user-supplied clock (50 MHz max.) in place of the internal 25 MHz oscillator. This feature allows synchronization of multiple devices. The PCA9635 does not have external clock input feature.
- Like the PCA9635, PCA9685 also has a built-in oscillator for the PWM control. However, the frequency used for PWM control in the PCA9685 is adjustable from about 24 Hz to 1526 Hz as compared to the typical 97.6 kHz frequency of the PCA9635. This allows the use of PCA9685 with external power supply controllers. All bits are set at the same frequency.
- The Power-On Reset (POR) default state of LEDn output pins is LOW in the case of PCA9685. It is HIGH for PCA9635.

The active LOW Output Enable input pin ($\overline{OE}$) allows asynchronous control of the LED outputs and can be used to set all the outputs to a defined I²C-bus programmable logic state. The $\overline{OE}$ can also be used to externally 'pulse width modulate' the outputs, which is useful when multiple devices need to be dimmed or blinked together using software control.

Software programmable LED All Call and three Sub Call I²C-bus addresses allow all or defined groups of PCA9685 devices to respond to a common I²C-bus address, allowing for example, all red LEDs to be turned on or off at the same time or marquee chasing effect, thus minimizing I²C-bus commands. Six hardware address pins allow up to 62 devices on the same bus.

The Software Reset (SWRST) General Call allows the master to perform a reset of the PCA9685 through the I²C-bus, identical to the Power-On Reset (POR) that initializes the registers to their default state causing the outputs to be set LOW. This allows an easy and quick way to reconfigure all device registers to the same condition via software.

2. Features and benefits

- 16 LED drivers. Each output programmable at:
  - Off
  - On
  - Programmable LED brightness
  - Programmable LED turn-on time to help reduce EMI
- 1 MHz Fast-mode Plus compatible I²C-bus interface with 30 mA high drive capability on SDA output for driving high capacitive buses
- 4096-step (12-bit) linear programmable brightness per LED output varying from fully off (default) to maximum brightness
- LED output frequency (all LEDs) typically varies from 24 Hz to 1526 Hz (Default of 1Eh in PRE_SCALE register results in a 200 Hz refresh rate with oscillator clock of 25 MHz.)
- Sixteen totem pole outputs (sink 25 mA and source 10 mA at 5 V) with software programmable open-drain LED outputs selection (default at totem pole). No input function.
- Output state change programmable on the Acknowledge or the STOP Command to update outputs byte-by-byte or all at the same time (default to 'Change on STOP').
- Active LOW Output Enable ($\overline{OE}$) input pin. LEDn outputs programmable to logic 1, logic 0 (default at power-up) or 'high-impedance' when $\overline{OE}$ is HIGH.
- 6 hardware address pins allow 62 PCA9685 devices to be connected to the same I²C-bus
- Toggling $\overline{OE}$ allows for hardware LED blinking
- 4 software programmable I²C-bus addresses (one LED All Call address and three LED Sub Call addresses) allow groups of devices to be addressed at the same time in any combination (for example, one register used for 'All Call' so that all the PCA9685s on the I²C-bus can be addressed at the same time and the second register used for three different addresses so that $1/3$ of all devices on the bus can be addressed at the same time in a group). Software enable and disable for these I²C-bus address.
- Software Reset feature (SWRST General Call) allows the device to be reset through the I²C-bus

- 25 MHz typical internal oscillator requires no external components
- External 50 MHz (max.) clock input
- Internal power-on reset
- Noise filter on SDA/SCL inputs
- Edge rate control on outputs
- No output glitches on power-up
- Supports hot insertion
- Low standby current
- Operating power supply voltage range of 2.3 V to 5.5 V
- 5.5 V tolerant inputs
- −40 °C to +85 °C operation
- ESD protection exceeds 2000 V HBM per JESD22-A114, 200 V MM per JESD22-A115 and 1000 V CDM per JESD22-C101
- Latch-up testing is done to JEDEC Standard JESD78 which exceeds 100 mA
- Packages offered: TSSOP28, HVQFN28

3. Applications

- RGB or RGBA LED drivers
- LED status information
- LED displays
- LCD backlights
- Keypad backlights for cellular phones or handheld devices

4. Ordering information

Table 1. Ordering information

Type number	Topside mark	Package		
		Name	Description	Version
PCA9685PW	PCA9685PW	TSSOP28	plastic thin shrink small outline package; 28 leads; body width 4.4 mm	SOT361-1
PCA9685PW/Q900[1]	PCA9685PW	TSSOP28	plastic thin shrink small outline package; 28 leads; body width 4.4 mm	SOT361-1
PCA9685BS	P9685	HVQFN28	plastic thermal enhanced very thin quad flat package; no leads; 28 terminals; body 6 × 6 × 0.85 mm	SOT788-1

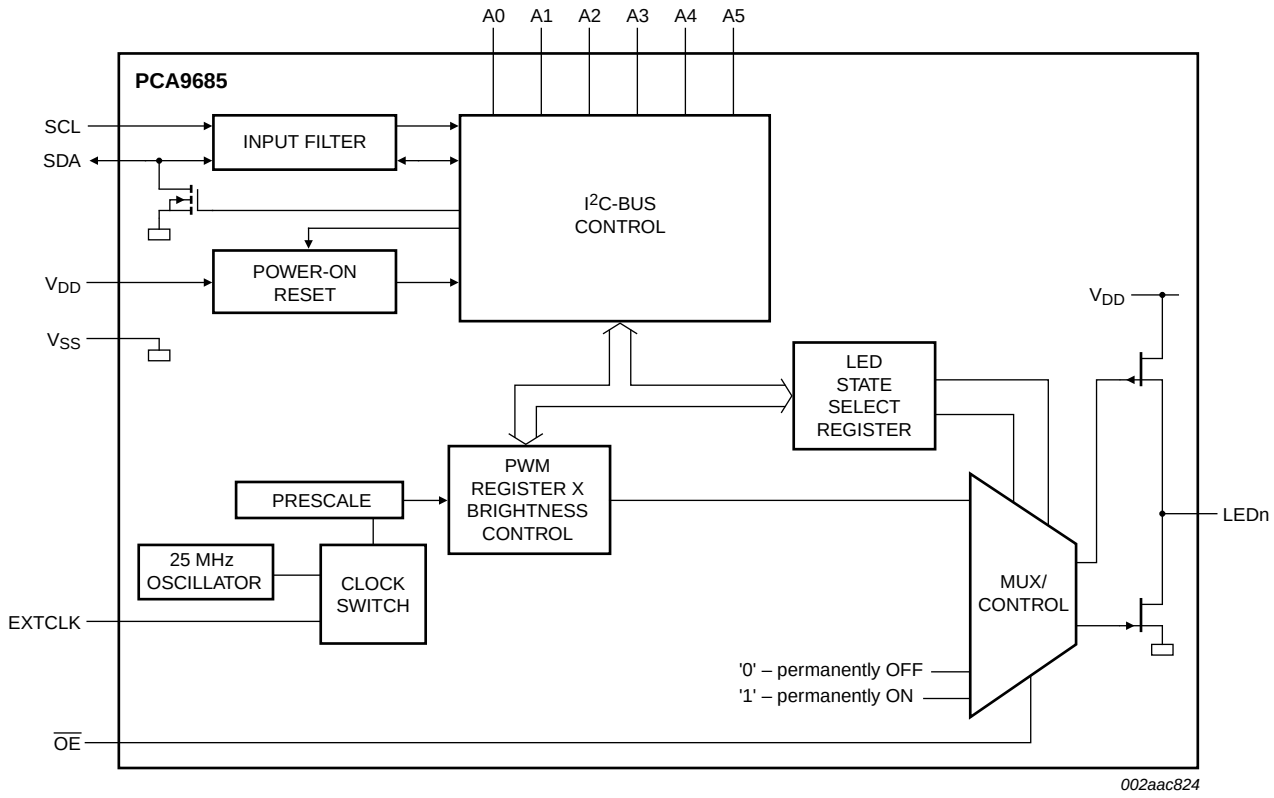
[1] PCA9685PW/Q900 is AEC-Q100 compliant. Contact **i2c.support@nxp.com** for PPAP.

4.1 Ordering options

Table 2. Ordering options

Type number	Orderable part number	Package	Packing method	Minimum order quantity	Temperature
PCA9685PW	PCA9685PW,118	TSSOP28	REEL 13" Q1/T1 *STANDARD MARK SMD	2500	T_{amb} = −40 °C to +85 °C
PCA9685PW/Q900	PCA9685PW/Q900,118	TSSOP28	REEL 13" Q1/T1 *STANDARD MARK SMD	2500	T_{amb} = −40 °C to +85 °C
PCA9685BS	PCA9685BS,118	HVQFN28	REEL 13" Q1/T1 *STANDARD MARK SMD	4000	T_{amb} = −40 °C to +85 °C

5. Block diagram



Remark: Only one LED output shown for clarity.

Fig 1. Block diagram of PCA9685

6. Pinning information

6.1 Pinning

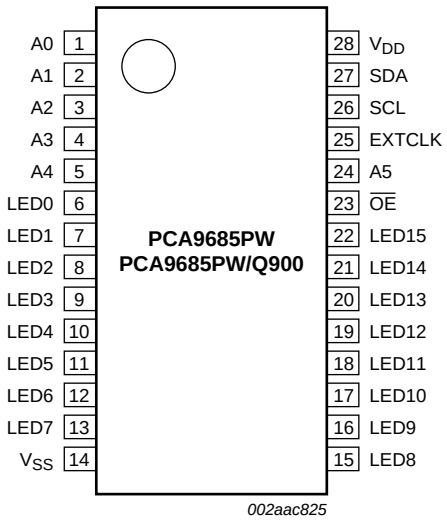


Fig 2. Pin configuration for TSSOP28

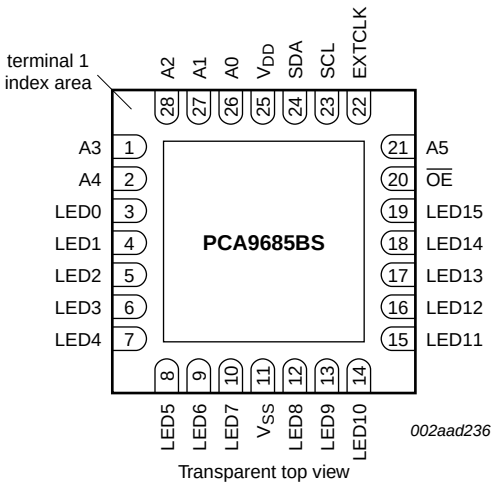


Fig 3. Pin configuration for HVQFN28

6.2 Pin description

Table 3. Pin description

Symbol	Pin		Type	Description
	TSSOP28	HVQFN28		
A0	1	26	I	address input 0
A1	2	27	I	address input 1
A2	3	28	I	address input 2
A3	4	1	I	address input 3
A4	5	2	I	address input 4
LED0	6	3	O	LED driver 0
LED1	7	4	O	LED driver 1
LED2	8	5	O	LED driver 2
LED3	9	6	O	LED driver 3
LED4	10	7	O	LED driver 4
LED5	11	8	O	LED driver 5
LED6	12	9	O	LED driver 6
LED7	13	10	O	LED driver 7
V_{SS}	14	11[1]	power supply	supply ground
LED8	15	12	O	LED driver 8
LED9	16	13	O	LED driver 9
LED10	17	14	O	LED driver 10
LED11	18	15	O	LED driver 11

Table 3. Pin description *...continued*

Symbol	Pin		Type	Description
	TSSOP28	HVQFN28		
LED12	19	16	O	LED driver 12
LED13	20	17	O	LED driver 13
LED14	21	18	O	LED driver 14
LED15	22	19	O	LED driver 15
$\overline{OE}$	23	20	I	active LOW output enable
A5	24	21	I	address input 5
EXTCLK	25	22	I	external clock input[2]
SCL	26	23	I	serial clock line
SDA	27	24	I/O	serial data line
V_{DD}	28	25	power supply	supply voltage

[1] HVQFN28 package die supply ground is connected to both V_{SS} pin and exposed center pad. V_{SS} pin must be connected to supply ground for proper device operation. For enhanced thermal, electrical, and board level performance, the exposed pad needs to be soldered to the board using a corresponding thermal pad on the board and for proper heat conduction through the board, thermal vias need to be incorporated in the PCB in the thermal pad region.

[2] This pin must be grounded when this feature is not used.

7. Functional description

Refer to Figure 1 “Block diagram of PCA9685”.

7.1 Device addresses

Following a START condition, the bus master must output the address of the slave it is accessing.

There are a maximum of 64 possible programmable addresses using the 6 hardware address pins. Two of these addresses, Software Reset and LED All Call, cannot be used because their default power-up state is ON, leaving a maximum of 62 addresses. Using other reserved addresses, as well as any other subcall address, will reduce the total number of possible addresses even further.

7.1.1 Regular I²C-bus slave address

The I²C-bus slave address of the PCA9685 is shown in Figure 4. To conserve power, no internal pull-up resistors are incorporated on the hardware selectable address pins and they must be pulled HIGH or LOW.

Remark: Using reserved I²C-bus addresses will interfere with other devices, but only if the devices are on the bus and/or the bus will be open to other I²C-bus systems at some later date. In a closed system where the designer controls the address assignment these addresses can be used since the PCA9685 treats them like any other address. The LED All Call, Software Reset and PCA9564 or PCA9665 slave address (if on the bus) can never be used for individual device addresses.

- PCA9685 LED All Call address (1110 000) and Software Reset (0000 0110) which are active on start-up

- PCA9564 (0000 000) or PCA9665 (1110 000) slave address which is active on start-up
- 'reserved for future use' I²C-bus addresses (0000 011, 1111 1XX)
- slave devices that use the 10-bit addressing scheme (1111 0XX)
- slave devices that are designed to respond to the General Call address (0000 000) which is used as the software reset address
- High-speed mode (Hs-mode) master code (0000 1XX)

Fig 4. Slave address

The last bit of the address byte defines the operation to be performed. When set to logic 1 a read is selected, while a logic 0 selects a write operation.

7.1.2 LED All Call I²C-bus address

- Default power-up value (ALLCALLADR register): E0h or 1110 000X
- Programmable through I²C-bus (volatile programming)
- At power-up, LED All Call I²C-bus address is enabled. PCA9685 sends an ACK when E0h (R/$\overline{W}$ = 0) or E1h (R/$\overline{W}$ = 1) is sent by the master.

See Section 7.3.7 "ALLCALLADR, LED All Call I²C-bus address" for more detail.

Remark: The default LED All Call I²C-bus address (E0h or 1110 000X) must not be used as a regular I²C-bus slave address since this address is enabled at power-up. All the PCA9685s on the I²C-bus will acknowledge the address if sent by the I²C-bus master.

7.1.3 LED Sub Call I²C-bus addresses

- 3 different I²C-bus addresses can be used
- Default power-up values:
  - SUBADR1 register: E2h or 1110 001X
  - SUBADR2 register: E4h or 1110 010X
  - SUBADR3 register: E8h or 1110 100X
- Programmable through I²C-bus (volatile programming)
- At power-up, Sub Call I²C-bus addresses are disabled. PCA9685 does not send an ACK when E2h (R/$\overline{W}$ = 0) or E3h (R/$\overline{W}$ = 1), E4h (R/$\overline{W}$ = 0) or E5h (R/$\overline{W}$ = 1), or E8h (R/$\overline{W}$ = 0) or E9h (R/$\overline{W}$ = 1) is sent by the master.

See Section 7.3.6 "SUBADR1 to SUBADR3, I²C-bus subaddress 1 to 3" for more detail.

Remark: The default LED Sub Call I²C-bus addresses may be used as regular I²C-bus slave addresses as long as they are disabled.

7.1.4 Software Reset I²C-bus address

The address shown in Figure 5 is used when a reset of the PCA9685 needs to be performed by the master. The Software Reset address (SWRST Call) must be used with R/$\overline{W}$ = logic 0. If R/$\overline{W}$ = logic 1, the PCA9685 does not acknowledge the SWRST. See Section 7.6 “Software reset” for more detail.

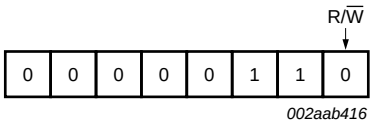


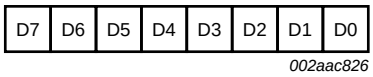
Fig 5. Software Reset address

Remark: The Software Reset I²C-bus address is a reserved address and cannot be used as a regular I²C-bus slave address or as an LED All Call or LED Sub Call address.

7.2 Control register

Following the successful acknowledgement of the slave address, LED All Call address or LED Sub Call address, the bus master will send a byte to the PCA9685, which will be stored in the Control register.

This register is used as a pointer to determine which register will be accessed.



reset state = 00h

Remark: The Control register does not apply to the Software Reset I²C-bus address.

Fig 6. Control register

7.3 Register definitions

Table 4. Register summary

Register # (decimal)	Register # (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
0	00	0	0	0	0	0	0	0	0	MODE1	read/write	Mode register 1
1	01	0	0	0	0	0	0	0	1	MODE2	read/write	Mode register 2
2	02	0	0	0	0	0	0	1	0	SUBADR1	read/write	I²C-bus subaddress 1
3	03	0	0	0	0	0	0	1	1	SUBADR2	read/write	I²C-bus subaddress 2
4	04	0	0	0	0	0	1	0	0	SUBADR3	read/write	I²C-bus subaddress 3
5	05	0	0	0	0	0	1	0	1	ALLCALLADR	read/write	LED All Call I²C-bus address
6	06	0	0	0	0	0	1	1	0	LED0_ON_L	read/write	LED0 output and brightness control byte 0
7	07	0	0	0	0	0	1	1	1	LED0_ON_H	read/write	LED0 output and brightness control byte 1
8	08	0	0	0	0	1	0	0	0	LED0_OFF_L	read/write	LED0 output and brightness control byte 2
9	09	0	0	0	0	1	0	0	1	LED0_OFF_H	read/write	LED0 output and brightness control byte 3
10	0A	0	0	0	0	1	0	1	0	LED1_ON_L	read/write	LED1 output and brightness control byte 0
11	0B	0	0	0	0	1	0	1	1	LED1_ON_H	read/write	LED1 output and brightness control byte 1
12	0C	0	0	0	0	1	1	0	0	LED1_OFF_L	read/write	LED1 output and brightness control byte 2
13	0D	0	0	0	0	1	1	0	1	LED1_OFF_H	read/write	LED1 output and brightness control byte 3
14	0E	0	0	0	0	1	1	1	0	LED2_ON_L	read/write	LED2 output and brightness control byte 0
15	0F	0	0	0	0	1	1	1	1	LED2_ON_H	read/write	LED2 output and brightness control byte 1
16	10	0	0	0	1	0	0	0	0	LED2_OFF_L	read/write	LED2 output and brightness control byte 2
17	11	0	0	0	1	0	0	0	1	LED2_OFF_H	read/write	LED2 output and brightness control byte 3
18	12	0	0	0	1	0	0	1	0	LED3_ON_L	read/write	LED3 output and brightness control byte 0
19	13	0	0	0	1	0	0	1	1	LED3_ON_H	read/write	LED3 output and brightness control byte 1
20	14	0	0	0	1	0	1	0	0	LED3_OFF_L	read/write	LED3 output and brightness control byte 2
21	15	0	0	0	1	0	1	0	1	LED3_OFF_H	read/write	LED3 output and brightness control byte 3

Table 4. Register summary *…continued*

Register # (decimal)	Register # (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
22	16	0	0	0	1	0	1	1	0	LED4_ON_L	read/write	LED4 output and brightness control byte 0
23	17	0	0	0	1	0	1	1	1	LED4_ON_H	read/write	LED4 output and brightness control byte 1
24	18	0	0	0	1	1	0	0	0	LED4_OFF_L	read/write	LED4 output and brightness control byte 2
25	19	0	0	0	1	1	0	0	1	LED4_OFF_H	read/write	LED4 output and brightness control byte 3
26	1A	0	0	0	1	1	0	1	0	LED5_ON_L	read/write	LED5 output and brightness control byte 0
27	1B	0	0	0	1	1	0	1	1	LED5_ON_H	read/write	LED5 output and brightness control byte 1
28	1C	0	0	0	1	1	1	0	0	LED5_OFF_L	read/write	LED5 output and brightness control byte 2
29	1D	0	0	0	1	1	1	0	1	LED5_OFF_H	read/write	LED5 output and brightness control byte 3
30	1E	0	0	0	1	1	1	1	0	LED6_ON_L	read/write	LED6 output and brightness control byte 0
31	1F	0	0	0	1	1	1	1	1	LED6_ON_H	read/write	LED6 output and brightness control byte 1
32	20	0	0	1	0	0	0	0	0	LED6_OFF_L	read/write	LED6 output and brightness control byte 2
33	21	0	0	1	0	0	0	0	1	LED6_OFF_H	read/write	LED6 output and brightness control byte 3
34	22	0	0	1	0	0	0	1	0	LED7_ON_L	read/write	LED7 output and brightness control byte 0
35	23	0	0	1	0	0	0	1	1	LED7_ON_H	read/write	LED7 output and brightness control byte 1
36	24	0	0	1	0	0	1	0	0	LED7_OFF_L	read/write	LED7 output and brightness control byte 2
37	25	0	0	1	0	0	1	0	1	LED7_OFF_H	read/write	LED7 output and brightness control byte 3
38	26	0	0	1	0	0	1	1	0	LED8_ON_L	read/write	LED8 output and brightness control byte 0
39	27	0	0	1	0	0	1	1	1	LED8_ON_H	read/write	LED8 output and brightness control byte 1
40	28	0	0	1	0	1	0	0	0	LED8_OFF_L	read/write	LED8 output and brightness control byte 2
41	29	0	0	1	0	1	0	0	1	LED8_OFF_H	read/write	LED8 output and brightness control byte 3

Table 4. Register summary *…continued*

Register # (decimal)	Register # (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
42	2A	0	0	1	0	1	0	1	0	LED9_ON_L	read/write	LED9 output and brightness control byte 0
43	2B	0	0	1	0	1	0	1	1	LED9_ON_H	read/write	LED9 output and brightness control byte 1
44	2C	0	0	1	0	1	1	0	0	LED9_OFF_L	read/write	LED9 output and brightness control byte 2
45	2D	0	0	1	0	1	1	0	1	LED9_OFF_H	read/write	LED9 output and brightness control byte 3
46	2E	0	0	1	0	1	1	1	0	LED10_ON_L	read/write	LED10 output and brightness control byte 0
47	2F	0	0	1	0	1	1	1	1	LED10_ON_H	read/write	LED10 output and brightness control byte 1
48	30	0	0	1	1	0	0	0	0	LED10_OFF_L	read/write	LED10 output and brightness control byte 2
49	31	0	0	1	1	0	0	0	1	LED10_OFF_H	read/write	LED10 output and brightness control byte 3
50	32	0	0	1	1	0	0	1	0	LED11_ON_L	read/write	LED11 output and brightness control byte 0
51	33	0	0	1	1	0	0	1	1	LED11_ON_H	read/write	LED11 output and brightness control byte 1
52	34	0	0	1	1	0	1	0	0	LED11_OFF_L	read/write	LED11 output and brightness control byte 2
53	35	0	0	1	1	0	1	0	1	LED11_OFF_H	read/write	LED11 output and brightness control byte 3
54	36	0	0	1	1	0	1	1	0	LED12_ON_L	read/write	LED12 output and brightness control byte 0
55	37	0	0	1	1	0	1	1	1	LED12_ON_H	read/write	LED12 output and brightness control byte 1
56	38	0	0	1	1	1	0	0	0	LED12_OFF_L	read/write	LED12 output and brightness control byte 2
57	39	0	0	1	1	1	0	0	1	LED12_OFF_H	read/write	LED12 output and brightness control byte 3
58	3A	0	0	1	1	1	0	1	0	LED13_ON_L	read/write	LED13 output and brightness control byte 0
59	3B	0	0	1	1	1	0	1	1	LED13_ON_H	read/write	LED13 output and brightness control byte 1
60	3C	0	0	1	1	1	1	0	0	LED13_OFF_L	read/write	LED13 output and brightness control byte 2
61	3D	0	0	1	1	1	1	0	1	LED13_OFF_H	read/write	LED13 output and brightness control byte 3

Table 4. Register summary *...continued*

Register # (decimal)	Register # (hex)	D7	D6	D5	D4	D3	D2	D1	D0	Name	Type	Function
62	3E	0	0	1	1	1	1	1	0	LED14_ON_L	read/write	LED14 output and brightness control byte 0
63	3F	0	0	1	1	1	1	1	1	LED14_ON_H	read/write	LED14 output and brightness control byte 1
64	40	0	1	0	0	0	0	0	0	LED14_OFF_L	read/write	LED14 output and brightness control byte 2
65	41	0	1	0	0	0	0	0	1	LED14_OFF_H	read/write	LED14 output and brightness control byte 3
66	42	0	1	0	0	0	0	1	0	LED15_ON_L	read/write	LED15 output and brightness control byte 0
67	43	0	1	0	0	0	0	1	1	LED15_ON_H	read/write	LED15 output and brightness control byte 1
68	44	0	1	0	0	0	1	0	0	LED15_OFF_L	read/write	LED15 output and brightness control byte 2
69	45	0	1	0	0	0	1	0	1	LED15_OFF_H	read/write	LED15 output and brightness control byte 3
...	reserved for future use											
250	FA	1	1	1	1	1	0	1	0	ALL_LED_ON_L	write/read zero	load all the LEDn_ON registers, byte 0
251	FB	1	1	1	1	1	0	1	1	ALL_LED_ON_H	write/read zero	load all the LEDn_ON registers, byte 1
252	FC	1	1	1	1	1	1	0	0	ALL_LED_OFF_L	write/read zero	load all the LEDn_OFF registers, byte 0
253	FD	1	1	1	1	1	1	0	1	ALL_LED_OFF_H	write/read zero	load all the LEDn_OFF registers, byte 1
254	FE	1	1	1	1	1	1	1	0	PRE_SCALE[1]	read/write	prescaler for PWM output frequency
255	FF	1	1	1	1	1	1	1	1	TestMode[2]	read/write	defines the test mode to be entered
...	All further addresses are reserved for future use; reserved addresses will not be acknowledged.											

[1] Writes to PRE_SCALE register are blocked when SLEEP bit is logic 0 (MODE 1).

[2] Reserved. Writes to this register may cause unpredictable results.

Remark: Auto Increment past register 69 will point to MODE1 register (register 0). Auto Increment also works from register 250 to register 254, then rolls over to register 0.

7.3.1 Mode register 1, MODE1

Table 5. MODE1 - Mode register 1 (address 00h) bit description

*Legend: * default value.*

Bit	Symbol	Access	Value	Description
7	RESTART	R		Shows state of RESTART logic. See Section 7.3.1.1 for detail.
		W		User writes logic 1 to this bit to clear it to logic 0. A user write of logic 0 will have no effect. See Section 7.3.1.1 for detail.
			0*	Restart disabled.
			1	Restart enabled.
6	EXTCLK	R/W		To use the EXTCLK pin, this bit must be set by the following sequence: 1. Set the SLEEP bit in MODE1. This turns off the internal oscillator. 2. Write logic 1s to both the SLEEP and EXTCLK bits in MODE1. The switch is now made. The external clock can be active during the switch because the SLEEP bit is set. This bit is a 'sticky bit', that is, it cannot be cleared by writing a logic 0 to it. The EXTCLK bit can **only** be cleared by a power cycle or software reset. EXTCLK range is DC to 50 MHz. $refresh\_rate = \frac{EXTCLK}{4096 \times (prescale + 1)}$
			0*	Use internal clock.
			1	Use EXTCLK pin clock.
5	AI	R/W	0*	Register Auto-Increment disabled[1].
			1	Register Auto-Increment enabled.
4	SLEEP	R/W	0	Normal mode[2].
			1*	Low power mode. Oscillator off[3][4].
3	SUB1	R/W	0*	PCA9685 does not respond to I²C-bus subaddress 1.
			1	PCA9685 responds to I²C-bus subaddress 1.
2	SUB2	R/W	0*	PCA9685 does not respond to I²C-bus subaddress 2.
			1	PCA9685 responds to I²C-bus subaddress 2.
1	SUB3	R/W	0*	PCA9685 does not respond to I²C-bus subaddress 3.
			1	PCA9685 responds to I²C-bus subaddress 3.
0	ALLCALL	R/W	0	PCA9685 does not respond to LED All Call I²C-bus address.
			1*	PCA9685 responds to LED All Call I²C-bus address.

[1] When the Auto Increment flag is set, AI = 1, the Control register is automatically incremented after a read or write. This allows the user to program the registers sequentially.

[2] It takes 500 μs max. for the oscillator to be up and running once SLEEP bit has been set to logic 0. Timings on LEDn outputs are not guaranteed if PWM control registers are accessed within the 500 μs window. There is no start-up delay required when using the EXTCLK pin as the PWM clock.

[3] No PWM control is possible when the oscillator is off.

[4] When the oscillator is off (Sleep mode) the LEDn outputs cannot be turned on, off or dimmed/blinked.

7.3.1.1 Restart mode

If the PCA9685 is operating and the user decides to put the chip to sleep (setting MODE1 bit 4) without stopping any of the PWM channels, the RESTART bit (MODE1 bit 7) will be set to logic 1 at the end of the PWM refresh cycle. The contents of each PWM register are held valid when the clock is off.

To restart all of the previously active PWM channels with a few I²C-bus cycles do the following steps:

1. Read MODE1 register.
2. Check that bit 7 (RESTART) is a logic 1. If it is, clear bit 4 (SLEEP). Allow time for oscillator to stabilize (500 μs).
3. Write logic 1 to bit 7 of MODE1 register. All PWM channels will restart and the RESTART bit will clear.

Remark: The SLEEP bit **must** be logic 0 for at least 500 μs, before a logic 1 is written into the RESTART bit.

Other actions that will clear the RESTART bit are:

1. Power cycle.
2. I²C Software Reset command.
3. If the MODE2 OCH bit is logic 0, write to any PWM register then issue an I²C-bus STOP.
4. If the MODE2 OCH bit is logic 1, write to all four PWM registers in any PWM channel.

Likewise, if the user does an orderly shutdown[1] of all the PWM channels before setting the SLEEP bit, the RESTART bit will be cleared. If this is done the contents of all PWM registers are invalidated and must be reloaded before reuse.

An example of the use of the RESTART bit would be the restoring of a customer's laptop LCD backlight intensity coming out of Standby to the level it was before going into Standby.

1. Two methods can be used to do an orderly shutdown. The fastest is to write a logic 1 to bit 4 in register ALL_LED_OFF_H. The other method is to write logic 1 to bit 4 in each active PWM channel LEDn_OFF_H register.

7.3.2 Mode register 2, MODE2

Table 6. MODE2 - Mode register 2 (address 01h) bit description

*Legend: * default value.*

Bit	Symbol	Access	Value	Description
7 to 5	-	read only	000*	reserved
4	INVRT[1]	R/W	0*	Output logic state not inverted. Value to use when external driver used. Applicable when $\overline{OE}$ = 0.
			1	Output logic state inverted. Value to use when no external driver used. Applicable when $\overline{OE}$ = 0.
3	OCH	R/W	0*	Outputs change on STOP command[2].
			1	Outputs change on ACK[3].
2	OUTDRV[1]	R/W	0	The 16 LEDn outputs are configured with an open-drain structure.
			1*	The 16 LEDn outputs are configured with a totem pole structure.
1 to 0	OUTNE[1:0][4]	R/W	00*	When $\overline{OE}$ = 1 (output drivers not enabled), LEDn = 0.
			01	When $\overline{OE}$ = 1 (output drivers not enabled): LEDn = 1 when OUTDRV = 1 LEDn = high-impedance when OUTDRV = 0 (same as OUTNE[1:0] = 10)
			1X	When $\overline{OE}$ = 1 (output drivers not enabled), LEDn = high-impedance.

[1] See Section 7.7 “Using the PCA9685 with and without external drivers” for more details. Normal LEDs can be driven directly in either mode. Some newer LEDs include integrated Zener diodes to limit voltage transients, reduce EMI, protect the LEDs and these must be driven only in the open-drain mode to prevent overheating the IC. Power on reset default state of LEDn output pins is LOW.

[2] Change of the outputs at the STOP command allows synchronizing outputs of more than one PCA9685. Applicable to registers from 06h (LED0_ON_L) to 45h (LED15_OFF_H) only. 1 or more registers can be written, in any order, before STOP.

[3] Update on ACK requires **all** 4 PWM channel registers to be loaded before outputs will change on the **last** ACK.

[4] See Section 7.4 “Active LOW output enable input” for more details.

7.3.3 LED output and PWM control

The turn-on time of each LED driver output and the duty cycle of PWM can be controlled independently using the LEDn_ON and LEDn_OFF registers.

There will be two 12-bit registers per LED output. These registers will be programmed by the user. Both registers will hold a value from 0 to 4095. One 12-bit register will hold a value for the ON time and the other 12-bit register will hold the value for the OFF time. The ON and OFF times are compared with the value of a 12-bit counter that will be running continuously from 0000h to 0FFFh (0 to 4095 decimal).

Update on ACK requires all 4 PWM channel registers to be loaded before outputs will change on the last ACK.

The ON time, which is programmable, will be the time the LED output will be asserted and the OFF time, which is also programmable, will be the time when the LED output will be negated. In this way, the phase shift becomes completely programmable. The resolution for the phase shift is $^1/_{4096}$ of the target frequency. Table 7 lists these registers.

The following two examples illustrate how to calculate values to be loaded into these registers.

Example 1: (assumes that the LED0 output is used and (delay time) + (PWM duty cycle) ≤ 100 %)

Delay time = 10 %; PWM duty cycle = 20 % (LED on time = 20 %; LED off time = 80 %).

Delay time = 10 % = 409.6 ~ 410 counts = 19Ah.

Since the counter starts at 0 and ends at 4095, we will subtract 1, so delay time = 199h counts.

LED0_ON_H = 1h; LED0_ON_L = 99h (LED start turn on after this delay count to 409)

LED on time = 20 % = 819.2 ~ 819 counts.

LED off time = 4CCh (decimal 410 + 819 – 1 = 1228)

LED0_OFF_H = 4h; LED0_OFF_L = CCh (LED start turn off after this count to 1228)

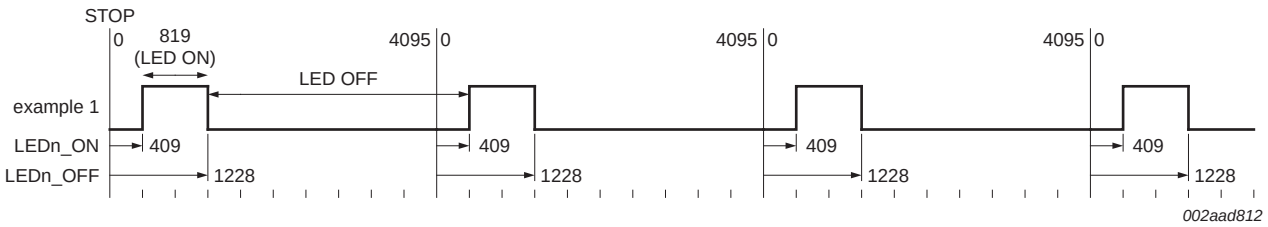


Fig 7. LED output, example 1

Example 2: (assumes that the LED4 output is used and (delay time) + (PWM duty cycle > 100 %)

Delay time = 90 %; PWM duty cycle = 90 % (LED on time = 90 %; LED off time = 10 %).

Delay time = 90 % = 3686.4 ~ 3686 counts – 1 = 3685 = E65h.

LED4_ON_H = Eh; LED4_ON_L = 65h (LED start turn on after this delay count to 3685)

LED on time = 90 % = 3686 counts.

Since the delay time and LED on period of the duty cycle is greater than 4096 counts, the LEDn_OFF count will occur in the next frame. Therefore, 4096 is subtracted from the LEDn_OFF count to get the correct LEDn_OFF count. See Figure 9, Figure 10 and Figure 11.

LED off time = CCBh (decimal 3685 + 3686 = 7372 – 4096 = 3275)

LED4_OFF_H = Ch; LED4_OFF_L = CBh (LED start turn off after this count to 3275)

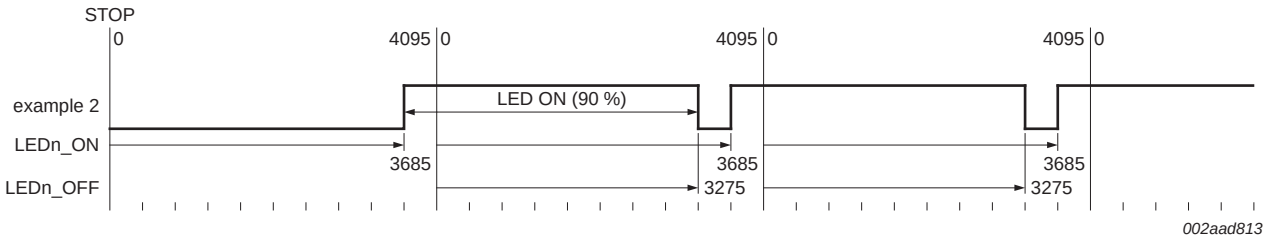
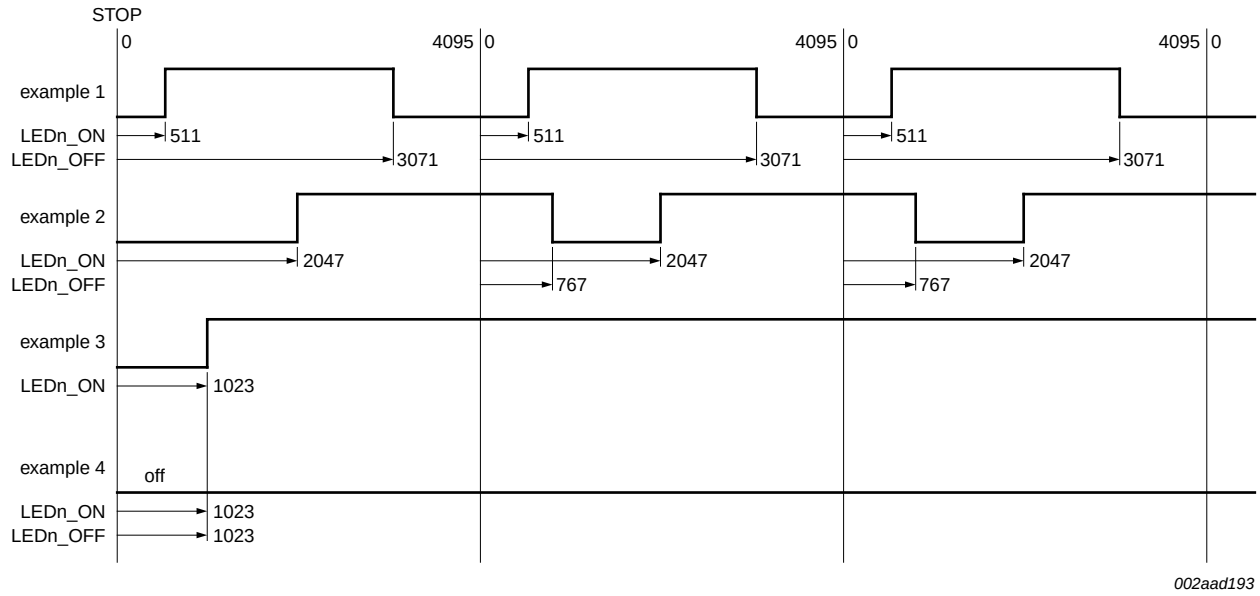


Fig 8. LED output, example 2



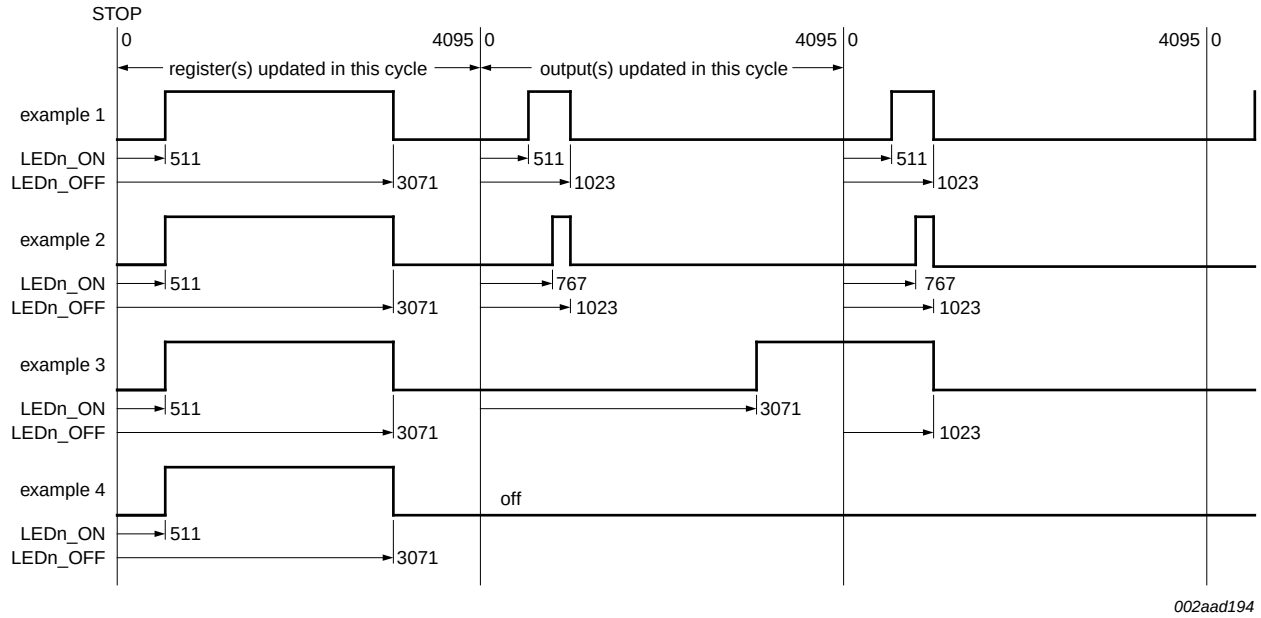
Example 1: LEDn_ON (511) < LEDn_OFF (3071)

Example 2: LEDn_ON (2047) > LEDn_OFF (767)

Example 3: LEDn_ON[12] = 1; LEDn_ON[11:0] = 1022; LEDn_OFF[12] = 0; LEDn_OFF[11:0] = don't care

Example 4: LEDn_ON[12] = 0; LEDn_OFF[12] = 0; LEDn_ON[11:0] = LEDn_OFF[11:0]

Fig 9. Output example



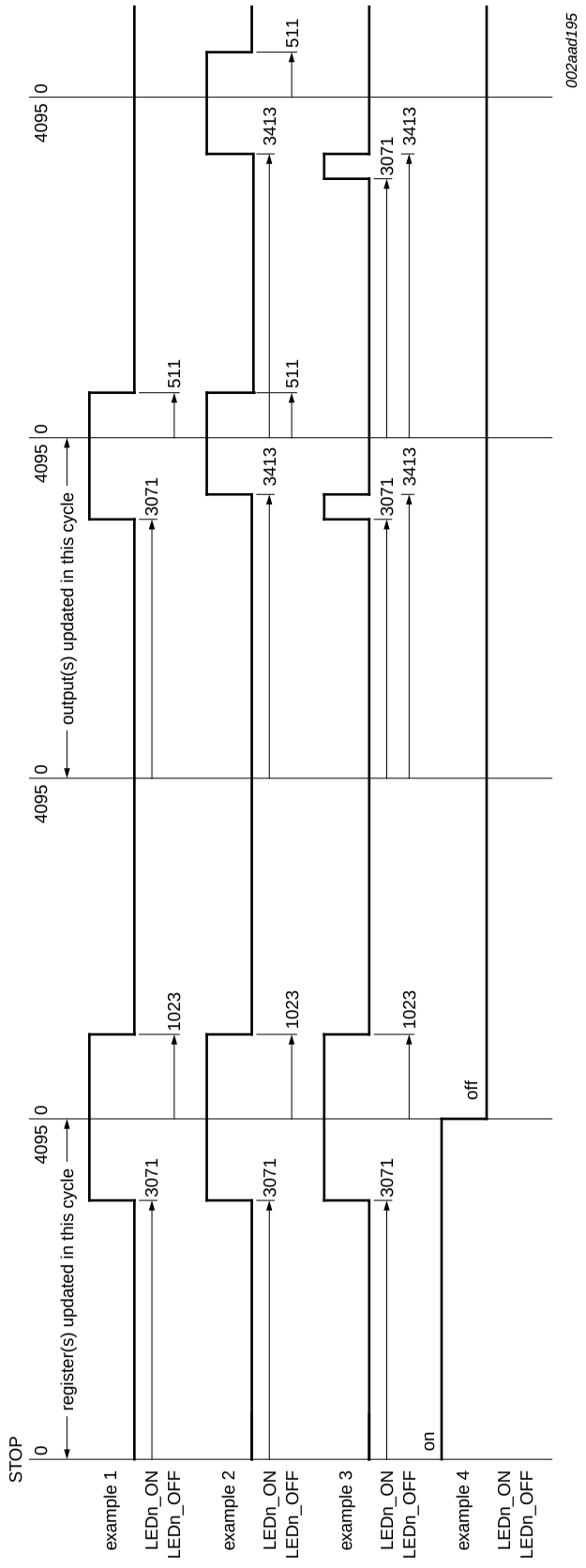
Example 1: LEDn_ON unchanged and LEDn_OFF decreased.

Example 2: LEDn_ON increased and LEDn_OFF decreased.

Example 3: LEDn_ON made > LEDn_OFF.

Example 4: LEDn_OFF[12] set to 1.

Fig 10. Update examples when LEDn_ON < LEDn_OFF



Example 1: LEDn_ON unchanged and LEDn_OFF decreased, but delay still > LEDn_OFF

Example 2: LEDn_ON changed and LEDn_OFF changed, but delay still > LEDn_OFF

Example 3: LEDn_ON unchanged and LEDn_OFF increased where LEDn_ON < LEDn_OFF

Example 4: LEDn_ON[12] = 1 and LEDn_OFF[12] changed from 0 to 1

Fig 11. Update examples when LEDn_ON > LEDn_OFF

Table 7. LED_ON, LED_OFF control registers (address 06h to 45h) bit description

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
06h	LED0_ON_L	7:0	LED0_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED0, 8 LSBs
07h	LED0_ON_H	7:5	reserved	R	000*	non-writable
		4	LED0_ON_H[4]	R/W	0*	LED0 full ON
		3:0	LED0_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED0, 4 MSBs
08h	LED0_OFF_L	7:0	LED0_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED0, 8 LSBs
09h	LED0_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED0_OFF_H[4]	R/W	1*	LED0 full OFF
		3:0	LED0_OFF_H[3:0]	R/W	0000*	
0Ah	LED1_ON_L	7:0	LED1_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED1, 8 LSBs
0Bh	LED1_ON_H	7:5	reserved	R	000*	non-writable
		4	LED1_ON_H[4]	R/W	0*	LED1 full ON
		3:0	LED1_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED1, 4 MSBs
0Ch	LED1_OFF_L	7:0	LED1_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED1, 8 LSBs
0Dh	LED1_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED1_OFF_H[4]	R/W	1*	LED1 full OFF
		3:0	LED1_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED1, 4 MSBs
0Eh	LED2_ON_L	7:0	LED2_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED2, 8 LSBs
0Fh	LED2_ON_H	7:5	reserved	R	000*	non-writable
		4	LED2_ON_H[4]	R/W	0*	LED2 full ON
		3:0	LED2_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED2, 4 MSBs
10h	LED2_OFF_L	7:0	LED2_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED2, 8 LSBs
11h	LED2_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED2_OFF_H[4]	R/W	1*	LED2 full OFF
		3:0	LED2_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED2, 4 MSBs
12h	LED3_ON_L	7:0	LED3_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED3, 8 LSBs
13h	LED3_ON_H	7:5	reserved	R	000*	non-writable
		4	LED3_ON_H[4]	R/W	0*	LED3 full ON
		3:0	LED3_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED3, 4 MSBs
14h	LED3_OFF_L	7:0	LED3_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED3, 8 LSBs
15h	LED3_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED3_OFF_H[4]	R/W	1*	LED3 full OFF
		3:0	LED3_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED3, 4 MSBs
16h	LED4_ON_L	7:0	LED4_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED4, 8 LSBs
17h	LED4_ON_H	7:5	reserved	R	000*	non-writable
		4	LED4_ON_H[4]	R/W	0*	LED4 full ON
		3:0	LED4_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED4, 4 MSBs

Table 7. LED_ON, LED_OFF control registers (address 06h to 45h) bit description *…continued*

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
18h	LED4_OFF_L	7:0	LED4_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED4, 8 LSBs
19h	LED4_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED4_OFF_H[4]	R/W	1*	LED4 full OFF
		3:0	LED4_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED4, 4 MSBs
1Ah	LED5_ON_L	7:0	LED5_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED5, 8 LSBs
1Bh	LED5_ON_H	7:5	reserved	R	000*	non-writable
		4	LED5_ON_H[4]	R/W	0*	LED5 full ON
		3:0	LED5_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED5, 4 MSBs
1Ch	LED5_OFF_L	7:0	LED5_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED5, 8 LSBs
1Dh	LED5_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED5_OFF_H[4]	R/W	1*	LED5 full OFF
		3:0	LED5_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED5, 4 MSBs
1Eh	LED6_ON_L	7:0	LED6_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED6, 8 LSBs
1Fh	LED6_ON_H	7:5	reserved	R	000*	non-writable
		4	LED6_ON_H[4]	R/W	0*	LED6 full ON
		3:0	LED6_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED6, 4 MSBs
20h	LED6_OFF_L	7:0	LED6_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED6, 8 LSBs
21h	LED6_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED6_OFF_H[4]	R/W	1*	LED6 full OFF
		3:0	LED6_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED6, 4 MSBs
22h	LED7_ON_L	7:0	LED7_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED7, 8 LSBs
23h	LED7_ON_H	7:5	reserved	R	000*	non-writable
		4	LED7_ON_H[4]	R/W	0*	LED7 full ON
		3:0	LED7_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED7, 4 MSBs
24h	LED7_OFF_L	7:0	LED7_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED7, 8 LSBs
25h	LED7_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED7_OFF_H[4]	R/W	1*	LED7 full OFF
		3:0	LED7_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED7, 4 MSBs
26h	LED8_ON_L	7:0	LED8_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED8, 8 LSBs
27h	LED8_ON_H	7:5	reserved	R	000*	non-writable
		4	LED8_ON_H[4]	R/W	0*	LED8 full ON
		3:0	LED8_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED8, 4 MSBs
28h	LED8_OFF_L	7:0	LED8_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED8, 8 LSBs
29h	LED8_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED8_OFF_H[4]	R/W	1*	LED8 full OFF
		3:0	LED8_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED8, 4 MSBs

Table 7. LED_ON, LED_OFF control registers (address 06h to 45h) bit description *…continued*

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
2Ah	LED9_ON_L	7:0	LED9_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED9, 8 LSBs
2Bh	LED9_ON_H	7:5	reserved	R	000*	non-writable
		4	LED9_ON_H[4]	R/W	0*	LED9 full ON
		3:0	LED9_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED9, 4 MSBs
2Ch	LED9_OFF_L	7:0	LED9_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED9, 8 LSBs
2Dh	LED9_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED9_OFF_H[4]	R/W	1*	LED9 full OFF
		3:0	LED9_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED9, 4 MSBs
2Eh	LED10_ON_L	7:0	LED10_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED10, 8 LSBs
2Fh	LED10_ON_H	7:5	reserved	R	000*	non-writable
		4	LED10_ON_H[4]	R/W	0*	LED10 full ON
		3:0	LED10_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED10, 4 MSBs
30h	LED10_OFF_L	7:0	LED10_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED10, 8 LSBs
31h	LED10_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED10_OFF_H[4]	R/W	1*	LED10 full OFF
		3:0	LED10_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED10, 4 MSBs
32h	LED11_ON_L	7:0	LED11_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED11, 8 LSBs
33h	LED11_ON_H	7:5	reserved	R	000*	non-writable
		4	LED11_ON_H[4]	R/W	0*	LED11 full ON
		3:0	LED11_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED11, 4 MSBs
34h	LED11_OFF_L	7:0	LED11_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED11, 8 LSBs
35h	LED11_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED11_OFF_H[4]	R/W	1*	LED11 full OFF
		3:0	LED11_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED11, 4 MSBs
36h	LED12_ON_L	7:0	LED12_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED12, 8 LSBs
37h	LED12_ON_H	7:5	reserved	R	000*	non-writable
		4	LED12_ON_H[4]	R/W	0*	LED12 full ON
		3:0	LED12_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED12, 4 MSBs
38h	LED12_OFF_L	7:0	LED12_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED12, 8 LSBs
39h	LED12_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED12_OFF_H[4]	R/W	1*	LED12 full OFF
		3:0	LED12_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED12, 4 MSBs
3Ah	LED13_ON_L	7:0	LED13_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED13, 8 LSBs
3Bh	LED13_ON_H	7:5	reserved	R	000*	non-writable
		4	LED13_ON_H[4]	R/W	0*	LED13 full ON
		3:0	LED13_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED13, 4 MSBs

Table 7. LED_ON, LED_OFF control registers (address 06h to 45h) bit description *…continued*

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
3Ch	LED13_OFF_L	7:0	LED13_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED13, 8 LSBs
3Dh	LED13_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED13_OFF_H[4]	R/W	1*	LED13 full OFF
		3:0	LED13_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED13, 4 MSBs
3Eh	LED14_ON_L	7:0	LED14_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED14, 8 LSBs
3Fh	LED14_ON_H	7:5	reserved	R	000*	non-writable
		4	LED14_ON_H[4]	R/W	0*	LED14 full ON
		3:0	LED14_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED14, 4 MSBs
40h	LED14_OFF_L	7:0	LED14_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED14, 8 LSBs
41h	LED14_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED14_OFF_H[4]	R/W	1*	LED14 full OFF
		3:0	LED14_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED14, 4 MSBs
42h	LED15_ON_L	7:0	LED15_ON_L[7:0]	R/W	0000 0000*	LEDn_ON count for LED15, 8 LSBs
43h	LED15_ON_H	7:5	reserved	R	000*	non-writable
		4	LED15_ON_H[4]	R/W	0*	LED15 full ON
		3:0	LED15_ON_H[3:0]	R/W	0000*	LEDn_ON count for LED15, 4 MSBs
44h	LED15_OFF_L	7:0	LED15_OFF_L[7:0]	R/W	0000 0000*	LEDn_OFF count for LED15, 8 LSBs
45h	LED15_OFF_H	7:5	reserved	R	000*	non-writable
		4	LED15_OFF_H[4]	R/W	1*	LED15 full OFF
		3:0	LED15_OFF_H[3:0]	R/W	0000*	LEDn_OFF count for LED15, 4 MSBs

The LEDn_ON_H output control bit 4, when set to logic 1, causes the output to be always ON. The turning ON of the LED is delayed by the amount in the LEDn_ON registers. LEDn_OFF[11:0] are ignored. When this bit = 0, then the LEDn_ON and LEDn_OFF registers are used according to their normal definition.

The LEDn_OFF_H output control bit 4, when set to logic 1, causes the output to be always OFF. In this case the values in the LEDn_ON registers are ignored.

Remark: When all LED outputs are configured as ‘always OFF’, the prescale counter and all associated PWM cycle timing logic are disabled. If LEDn_ON_H[4] and LEDn_OFF_H[4] are set at the same time, the LEDn_OFF_H[4] function takes precedence.

7.3.4 ALL_LED_ON and ALL_LED_OFF control

The ALL_LED_ON and ALL_LED_OFF registers allow just four I²C-bus write sequences to fill all the ON and OFF registers with the same patterns.

Table 8. ALL_LED_ON and ALL_LED_OFF control registers (address FAh to FEh) bit description

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
FAh	ALL_LED_ON_L	7:0	ALL_LED_ON_L[7:0]	W only	0000 0000*	LEDn_ON count for ALL_LED, 8 MSBs
FBh	ALL_LED_ON_H	7:5	reserved	R	000*	non-writable
		4	ALL_LED_ON_H[4]	W only	1*	ALL_LED full ON
		3:0	ALL_LED_ON_H[3:0]	W only	0000*	LEDn_ON count for ALL_LED, 4 MSBs
FCh	ALL_LED_OFF_L	7:0	ALL_LED_OFF_L[7:0]	W only	0000 0000*	LEDn_OFF count for ALL_LED, 8 MSBs
FDh	ALL_LED_OFF_H	7:5	reserved	R	000*	non-writable
		4	ALL_LED_OFF_H[4]	W only	1*	ALL_LED full OFF
		3:0	ALL_LED_OFF_H[3:0]	W only	0000*	LEDn_OFF count for ALL_LED, 4 MSBs
FEh	PRE_SCALE	7:0	PRE_SCALE[7:0]	R/W	0001 1110*	prescaler to program the PWM output frequency (default is 200 Hz)

The LEDn_ON and LEDn_OFF counts can vary from 0 to 4095. The LEDn_ON and LEDn_OFF count registers should never be programmed with the same values.

Because the loading of the LEDn_ON and LEDn_OFF registers is via the I²C-bus, and asynchronous to the internal oscillator, we want to ensure that we do not see any visual artifacts of changing the ON and OFF values. This is achieved by updating the changes at the end of the LOW cycle.

7.3.5 PWM frequency PRE_SCALE

The hardware forces a minimum value that can be loaded into the PRE_SCALE register at '3'. The PRE_SCALE register defines the frequency at which the outputs modulate. The prescale value is determined with the formula shown in Equation 1:

$$prescale\ value = round\left(\frac{osc\_clock}{4096 \times update\_rate}\right) - 1 \quad (1)$$

where the update rate is the output modulation frequency required. For example, for an output default frequency of 200 Hz with an oscillator clock frequency of 25 MHz:

$$prescale\ value = round\left(\frac{25\ MHz}{4096 \times 200}\right) - 1 = 30\ (0x1Eh) \quad (2)$$

The maximum PWM frequency is 1526 Hz if the PRE_SCALE register is set "0x03h".

The minimum PWM frequency is 24 Hz if the PRE_SCALE register is set "0xFFh".

The PRE_SCALE register can only be set when the SLEEP bit of MODE1 register is set to logic 1.

7.3.6 SUBADR1 to SUBADR3, I²C-bus subaddress 1 to 3

Table 9. SUBADR1 to SUBADR3 - I²C-bus subaddress registers 0 to 3 (address 02h to 04h) bit description

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
02h	SUBADR1	7:1	A1[7:1]	R/W	1110 001*	I²C-bus subaddress 1
		0	A1[0]	R only	0*	reserved
03h	SUBADR2	7:1	A2[7:1]	R/W	1110 010*	I²C-bus subaddress 2
		0	A2[0]	R only	0*	reserved
04h	SUBADR3	7:1	A3[7:1]	R/W	1110 100*	I²C-bus subaddress 3
		0	A3[0]	R only	0*	reserved

Subaddresses are programmable through the I²C-bus. Default power-up values are E2h, E4h, E8h, and the device(s) will not acknowledge these addresses right after power-up (the corresponding SUBx bit in MODE1 register is equal to 0).

Once subaddresses have been programmed to their right values, SUBx bits need to be set to logic 1 in order to have the device acknowledging these addresses (MODE1 register).

Only the 7 MSBs representing the I²C-bus subaddress are valid. The LSB in SUBADRx register is a read-only bit (0).

When SUBx is set to logic 1, the corresponding I²C-bus subaddress can be used during either an I²C-bus read or write sequence.

7.3.7 ALLCALLADR, LED All Call I²C-bus address

Table 10. ALLCALLADR - LED All Call I²C-bus address register (address 05h) bit description

*Legend: * default value.*

Address	Register	Bit	Symbol	Access	Value	Description
05h	ALLCALLADR	7:1	AC[7:1]	R/W	1110 000*	ALLCALL I²C-bus address register
		0	AC[0]	R only	0*	reserved

The LED All Call I²C-bus address allows all the PCA9685s in the bus to be programmed at the same time (ALLCALL bit in register MODE1 must be equal to 1 (power-up default state)). This address is programmable through the I²C-bus and can be used during either an I²C-bus read or write sequence. The register address can also be programmed as a Sub Call.

Only the 7 MSBs representing the All Call I²C-bus address are valid. The LSB in ALLCALLADR register is a read-only bit (0).

If ALLCALL bit = 0, the device does not acknowledge the address programmed in register ALLCALLADR.

7.4 Active LOW output enable input

The active LOW output enable ($\overline{OE}$) pin, allows to enable or disable all the LED outputs at the same time.

- When a LOW level is applied to $\overline{OE}$ pin, all the LED outputs are enabled and follow the output state defined in the LEDn_ON and LEDn_OFF registers with the polarity defined by INVRT bit (MODE2 register).
- When a HIGH level is applied to $\overline{OE}$ pin, all the LED outputs are programmed to the value that is defined by OUTNE[1:0] in the MODE2 register.

Table 11. LED outputs when $\overline{OE}$ = 1

OUTNE1	OUTNE0	LED outputs
0	0	0
0	1	1 if OUTDRV = 1, high-impedance if OUTDRV = 0
1	0	high-impedance
1	1	high-impedance

The $\overline{OE}$ pin can be used as a synchronization signal to switch on/off several PCA9685 devices at the same time. This requires an external clock reference that provides blinking period and the duty cycle.

The $\overline{OE}$ pin can also be used as an external dimming control signal. The frequency of the external clock must be high enough not to be seen by the human eye, and the duty cycle value determines the brightness of the LEDs.

7.5 Power-on reset

When power is applied to V_{DD}, an internal power-on reset holds the PCA9685 in a reset condition until V_{DD} has reached V_{POR}. At this point, the reset condition is released and the PCA9685 registers and I²C-bus state machine are initialized to their default states. Thereafter, V_{DD} must be lowered below 0.2 V to reset the device.

7.6 Software reset

The Software Reset Call (SWRST Call) allows all the devices in the I²C-bus to be reset to the power-up state value through a specific formatted I²C-bus command. To be performed correctly, it implies that the I²C-bus is functional and that there is no device hanging the bus.

The SWRST Call function is defined as the following:

1. A START command is sent by the I²C-bus master.
2. The reserved SWRST I²C-bus address '0000 000' with the R/$\overline{W}$ bit set to '0' (write) is sent by the I²C-bus master.
3. The PCA9685 device(s) acknowledge(s) after seeing the General Call address '0000 0000' (00h) only. If the R/$\overline{W}$ bit is set to '1' (read), no acknowledge is returned to the I²C-bus master.
4. Once the General Call address has been sent and acknowledged, the master sends 1 byte with 1 specific value (SWRST data byte 1):
   a. Byte 1 = 06h: the PCA9685 acknowledges this value only. If byte 1 is not equal to 06h, the PCA9685 does not acknowledge it.

   If more than 1 byte of data is sent, the PCA9685 does not acknowledge any more.
5. Once the correct byte (SWRST data byte 1) has been sent and correctly acknowledged, the master sends a STOP command to end the SWRST Call: the PCA9685 then resets to the default value (power-up value) and is ready to be addressed again within the specified bus free time (t_{BUF}).

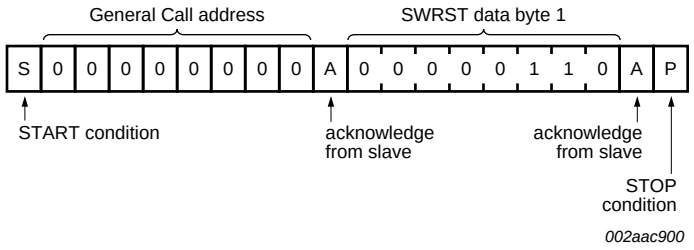


Fig 12. SWRST Call

The I²C-bus master must interpret a non-acknowledge from the PCA9685 (at any time) as a 'SWRST Call Abort'. The PCA9685 does not initiate a reset of its registers. This happens only when the format of the SWRST Call sequence is not correct.

7.7 Using the PCA9685 with and without external drivers

The PCA9685 LED output drivers are 5.5 V only tolerant and can sink up to 25 mA at 5 V.

If the device needs to drive LEDs to a higher voltage and/or higher current, use of an external driver is required.

- INVRT bit (MODE2 register) can be used to keep the LED PWM control firmware the same independently of the type of external driver. This bit allows LED output polarity inversion/non-inversion only when $\overline{OE}$ = 0.
- OUTDRV bit (MODE2 register) allows minimizing the amount of external components required to control the external driver (N-type or P-type device).

Table 12. Use of INVRT and OUTDRV based on connection to the LEDn outputs when $\overline{OE}$ = 0[1]

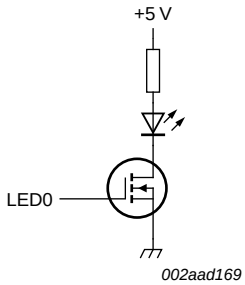
INVRT	OUTDRV	Direct connection to LEDn		External N-type driver		External P-type driver	
		Firmware	External pull-up resistor	Firmware	External pull-up resistor	Firmware	External pull-up resistor
0	0	formulas and LED output state values inverted	LED current limiting R[2]	formulas and LED output state values inverted	required	formulas and LED output state values apply	required
0	1	formulas and LED output state values inverted	LED current limiting R[2]	formulas and LED output state values apply[3]	not required[3]	formulas and LED output state values inverted	not required
1	0	formulas and LED output state values apply[2]	LED current limiting R	formulas and LED output state values apply	required	formulas and LED output state values inverted	required
1	1	formulas and LED output state values apply[2]	LED current limiting R	formulas and LED output state values inverted	not required	formulas and LED output state values apply[4]	not required[4]

[1] When $\overline{OE}$ = 1, LED output state is controlled only by OUTNE[1:0] bits (MODE2 register).

[2] Correct configuration when LEDs directly connected to the LEDn outputs (connection to V_{DD} through current limiting resistor).

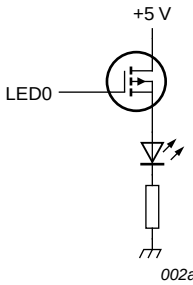
[3] Optimum configuration when external N-type (NPN, NMOS) driver used.

[4] Optimum configuration when external P-type (PNP, PMOS) driver used.



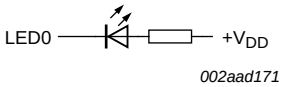
INVRT = 0
OUTDRV = 1

Fig 13. External N-type driver



INVRT = 1
OUTDRV = 1

Fig 14. External P-type driver



INVRT = 1
OUTDRV = 0

Fig 15. Direct LED connection

8. Characteristics of the I²C-bus

The I²C-bus is for 2-way, 2-line communication between different ICs or modules. The two lines are a serial data line (SDA) and a serial clock line (SCL). Both lines must be connected to a positive supply via a pull-up resistor when connected to the output stages of a device. Data transfer may be initiated only when the bus is not busy.

8.1 Bit transfer

One data bit is transferred during each clock pulse. The data on the SDA line must remain stable during the HIGH period of the clock pulse as changes in the data line at this time will be interpreted as control signals (see Figure 16).

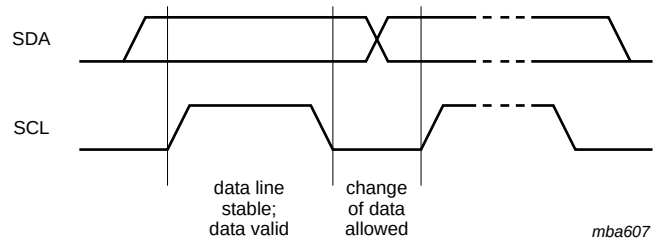


Fig 16. Bit transfer

8.1.1 START and STOP conditions

Both data and clock lines remain HIGH when the bus is not busy. A HIGH-to-LOW transition of the data line while the clock is HIGH is defined as the START condition (S). A LOW-to-HIGH transition of the data line while the clock is HIGH is defined as the STOP condition (P) (see Figure 17).

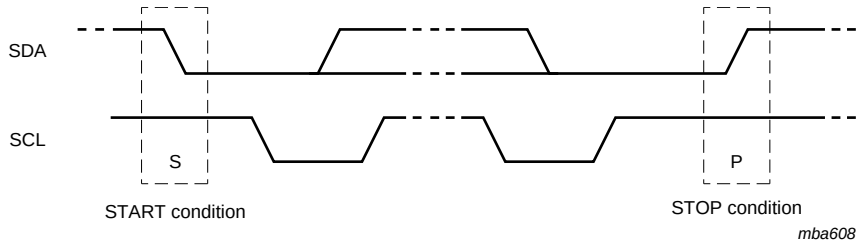


Fig 17. Definition of START and STOP conditions

8.2 System configuration

A device generating a message is a ‘transmitter’; a device receiving is the ‘receiver’. The device that controls the message is the ‘master’ and the devices which are controlled by the master are the ‘slaves’ (see Figure 18).

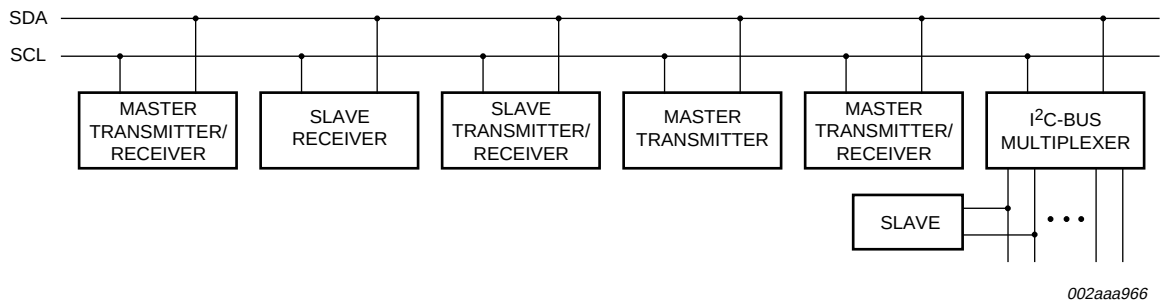


Fig 18. System configuration

8.3 Acknowledge

The number of data bytes transferred between the START and the STOP conditions from transmitter to receiver is not limited. Each byte of eight bits is followed by one acknowledge bit. The acknowledge bit is a HIGH level put on the bus by the transmitter, whereas the master generates an extra acknowledge related clock pulse.

A slave receiver which is addressed must generate an acknowledge after the reception of each byte. Also a master must generate an acknowledge after the reception of each byte that has been clocked out of the slave transmitter. The device that acknowledges has to pull down the SDA line during the acknowledge clock pulse, so that the SDA line is stable LOW during the HIGH period of the acknowledge related clock pulse; set-up time and hold time must be taken into account.

A master receiver must signal an end of data to the transmitter by not generating an acknowledge on the last byte that has been clocked out of the slave. In this event, the transmitter must leave the data line HIGH to enable the master to generate a STOP condition.

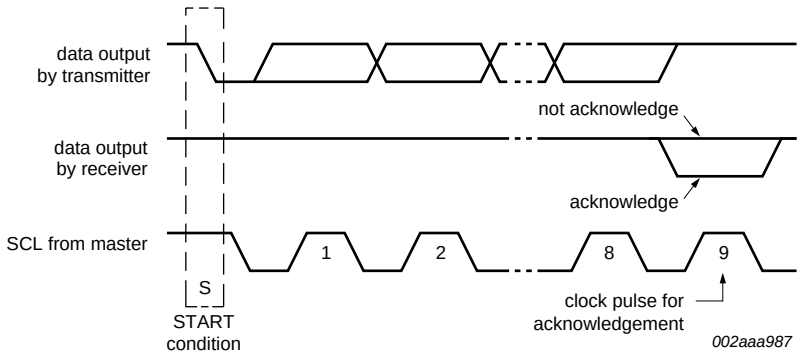
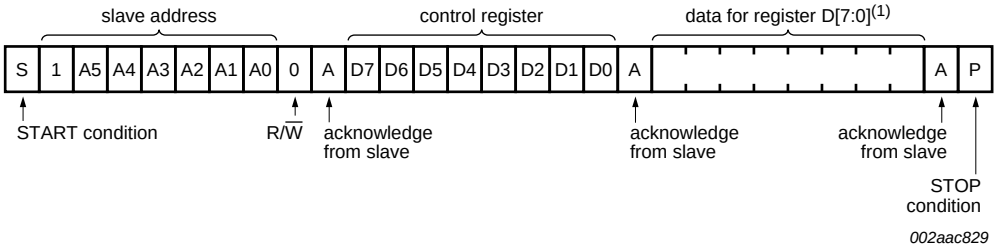


Fig 19. Acknowledgement on the I²C-bus

9. Bus transactions



(1) See Table 4 for register definition.

Fig 20. Write to a specific register

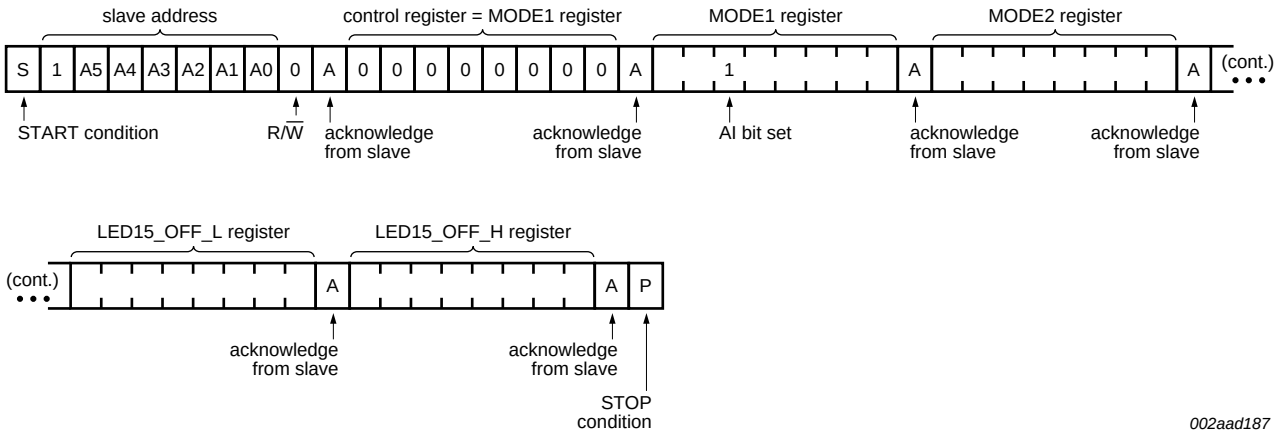


Fig 21. Write to all registers using the Auto-Increment feature; AI initially clear

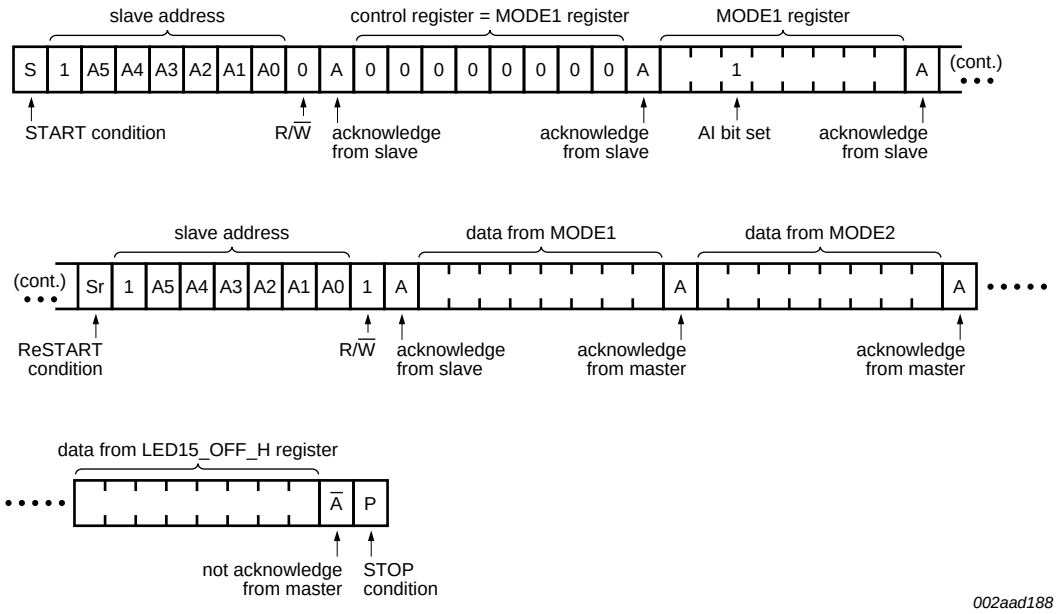


Fig 22. Read all registers using the Auto-Increment feature; AI initially clear

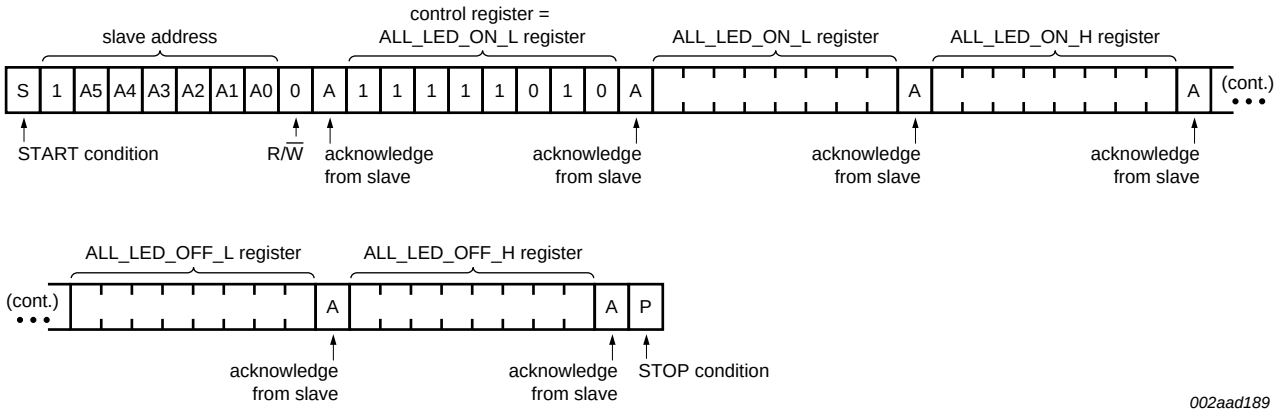


Fig 23. Write to ALL_LED_ON and ALL_LED_OFF registers using the Auto-Increment feature; AI initially set

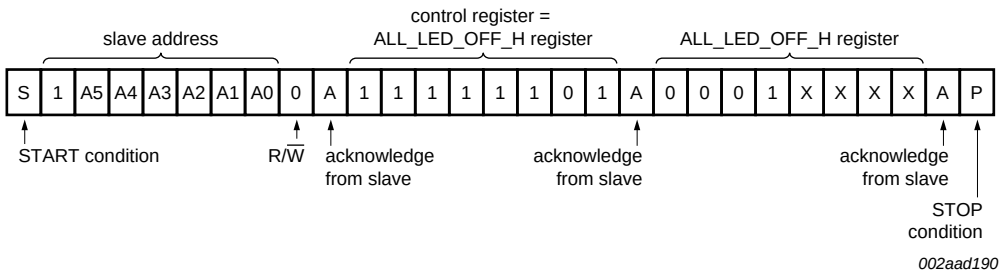
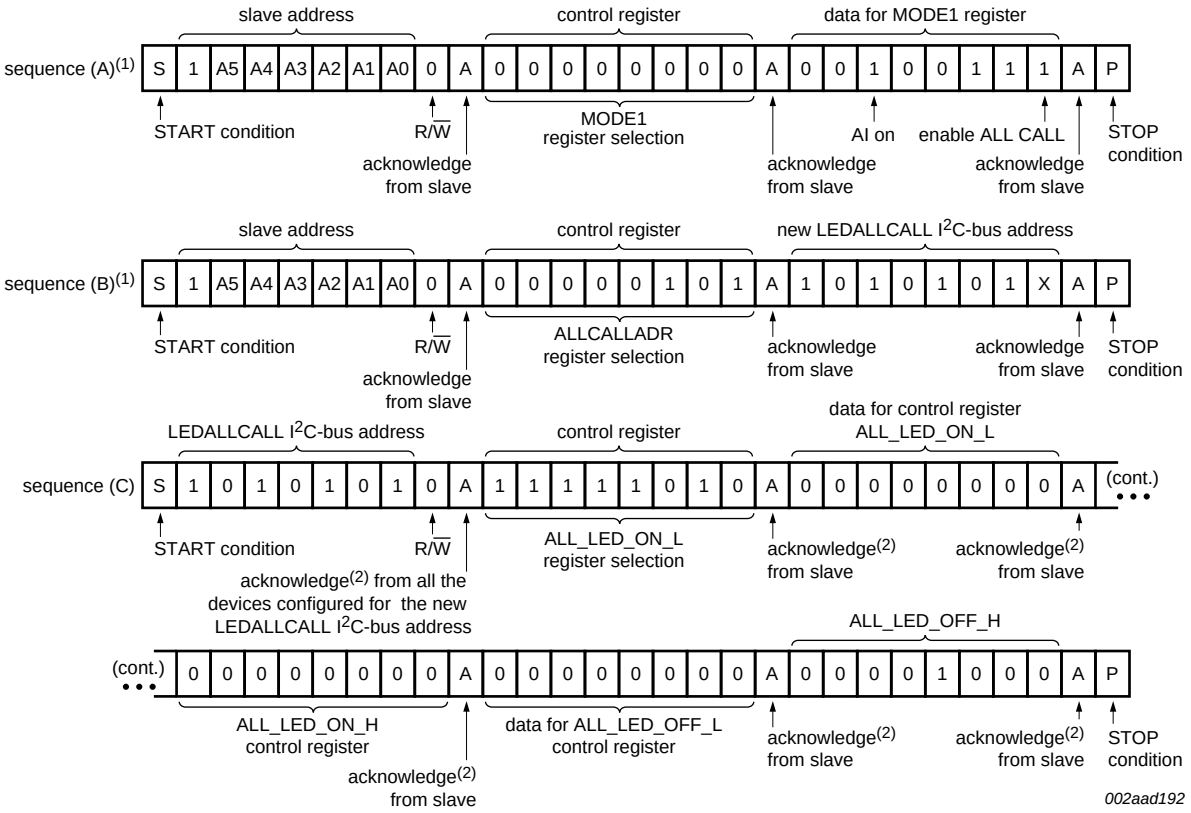


Fig 24. Write to ALL_LED_OFF_H to turn OFF all PWMs

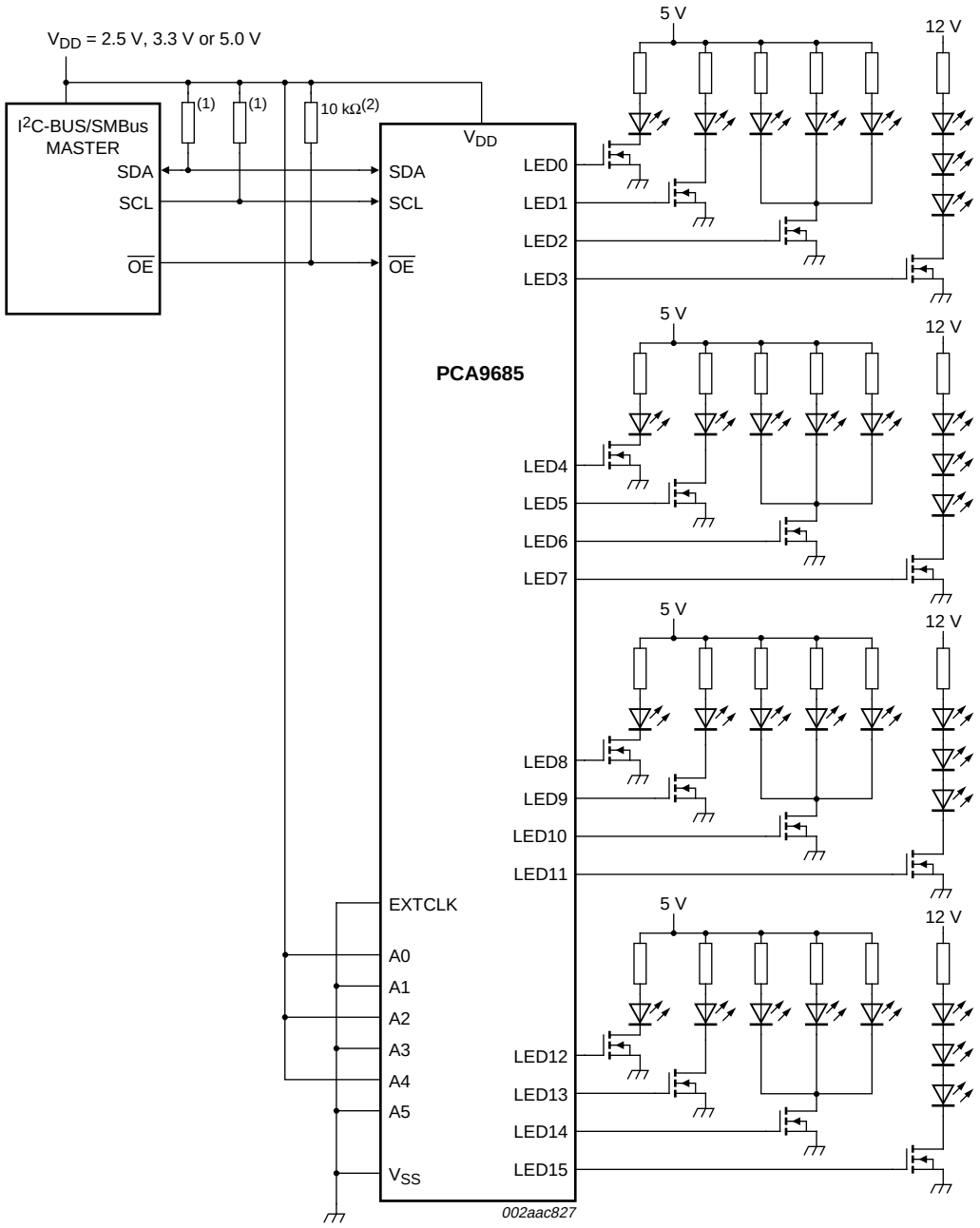


(1) In this example, several PCA9685s are used and the same sequences (A) and (B) above are sent to each of them.

(2) Acknowledge from all the slave devices configured for the new LED All Call I²C-bus address in sequence (B).

Fig 25. LED All Call I²C-bus address programming and LED All Call sequence example

10. Application design-in information



I²C-bus address = 1010 101x.

All 16 of the LEDn outputs configurable as either open-drain or totem pole. Mixing of configuration is not possible.

Remark: Set INVRT = 0, OUTDRV = 1, OUTNE = 01 (MODE2 register bits)

(1) Resistor value should be chosen by referencing section 7 of *UM10204, "I²C-bus specification and user manual"*.

(2) $\overline{OE}$ requires pull-up resistor if control signal from the master is open-drain.

Fig 26. Typical application

Question 1: What kind of edge rate control is there on the outputs?

- The typical edge rates depend on the output configuration, supply voltage, and the applied load. The outputs can be configured as either open-drain NMOS or totem pole outputs. If the customer is using the part to directly drive LEDs, they should be using it in an open-drain NMOS, if they are concerned about the maximum I_{SS} and ground bounce. The edge rate control was designed primarily to slow down the turn-on of the output device; it turns off rather quickly (~1.5 ns). In simulation, the typical turn-on time for the open-drain NMOS was ~14 ns (V_{DD} = 3.6 V; C_L = 50 pF; R_{PU} = 500 Ω).

Question 2: Is ground bounce possible?

- Ground bounce is a possibility, especially if all 16 outputs are changed at full current (25 mA each). There is a fair amount of decoupling capacitance on chip (~50 pF), which is intended to suppress some of the ground bounce. The customer will need to determine if additional decoupling capacitance externally placed as close as physically possible to the device is required.

Question 3: Can I really sink 400 mA through the single ground pin on the package and will this cause any ground bounce problem due to the PWM of the LEDs?

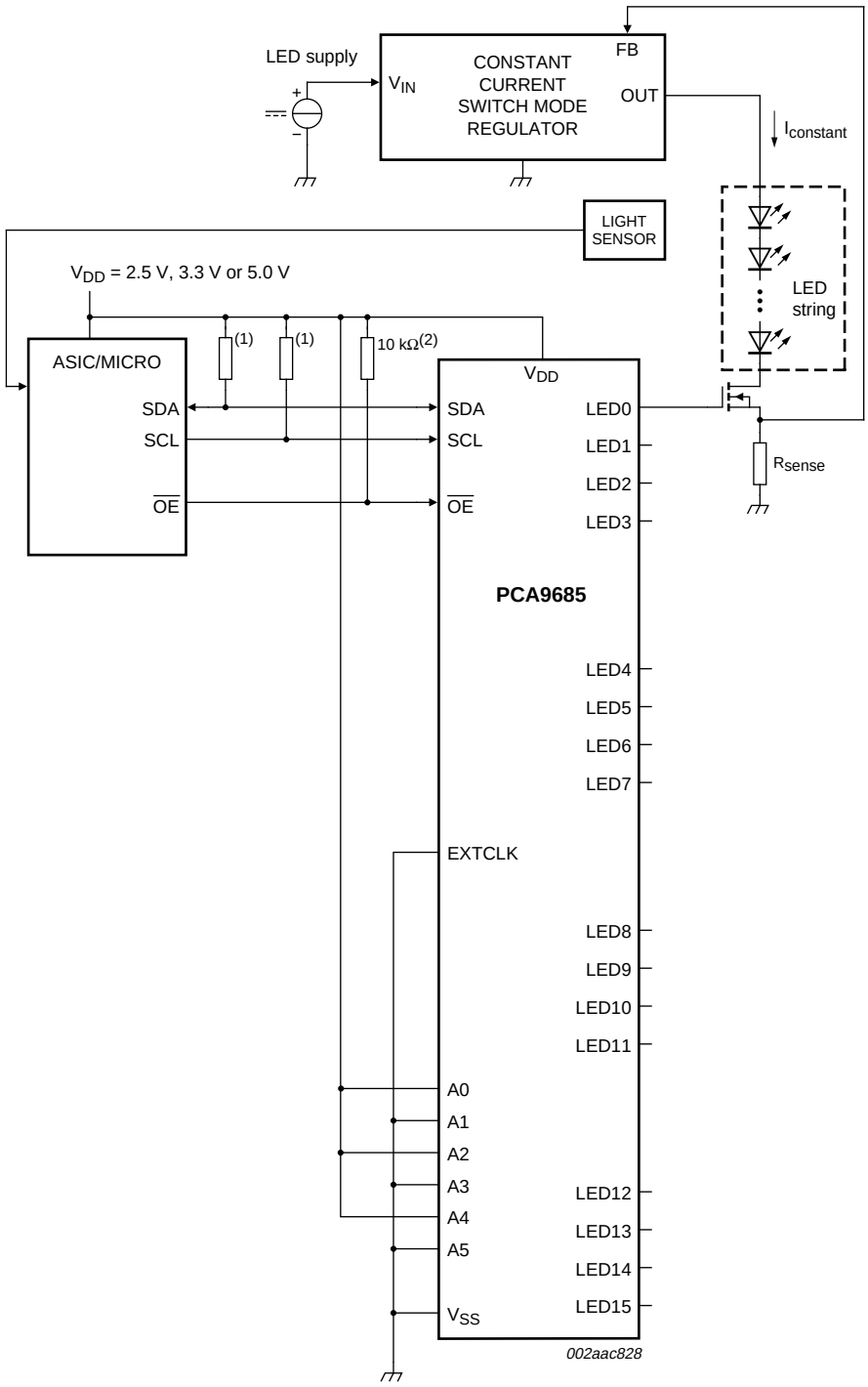
- Yes, you can sink 400 mA through a single ground pin on the **package**. Although the package only has one ground pin, there are two ground pads on the die itself connected to this one pin. Although some ground bounce is likely, it will not disrupt the operation of the part and would be reduced by the external decoupling capacitance.

Question 4: I can't turn the LEDs on or off, but their registers are set properly. Why?

- Check the MODE1 register SLEEP (bit 4) setting. The bit needs to be 0 in order to enable the clocking. If both clock sources (internal osc and EXTCLK) are turned OFF (bit 4 = 1), the LEDs cannot be dimmed or blinked.

Question 5: I'm using LEDs with integrated Zener diodes and the IC is getting very hot. Why?

- The IC outputs can be set to either open-drain or push-pull and default to push-pull outputs. In this application with the Zener diodes, they need to be set to open-drain since in the push-pull architecture there is a low resistance path to GND through the Zener and this is causing the IC to overheat.



I²C-bus address = 1010 101x.

Remark: Set INVRT = 0, OUTDRV = 1, OUTNE = 01 (MODE2 register bits) for this configuration.

(1) Resistor value should be chosen by referencing Section 7 of *UM10204, "I²C-bus specification and user manual"*.

(2) $\overline{OE}$ requires pull-up resistor if control signal from the master is open-drain.

Fig 27. LCD backlighting application

11. Limiting values

Table 13. Limiting values

In accordance with the Absolute Maximum Rating System (IEC 60134).

Symbol	Parameter	Conditions	Min	Max	Unit
V_{DD}	supply voltage		–0.5	+6.0	V
$V_{I/O}$	voltage on an input/output pin		V_{SS} – 0.5	5.5	V
$I_{O(LEDn)}$	output current on pin LEDn		-	25	mA
I_{SS}	ground supply current		-	400	mA
P_{tot}	total power dissipation		-	400	mW
T_{stg}	storage temperature		–65	+150	°C
T_{amb}	ambient temperature	operating	–40	+85	°C

12. Static characteristics

Table 14. Static characteristics

V_{DD} = 2.3 V to 5.5 V; V_{SS} = 0 V; T_{amb} = –40 °C to +85 °C; unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Supply						
V_{DD}	supply voltage		2.3	-	5.5	V
I_{DD}	supply current	operating mode; no load; f_{SCL} = 1 MHz; V_{DD} = 2.3 V to 5.5 V	-	6	10	mA
I_{stb}	standby current	no load; f_{SCL} = 0 Hz; V_I = V_{DD} or V_{SS}; V_{DD} = 2.3 V to 5.5 V	-	2.2	15.5	µA
V_{POR}	power-on reset voltage	no load; V_I = V_{DD} or V_{SS} [1]	-	1.70	2.0	V
Input SCL; input/output SDA						
V_{IL}	LOW-level input voltage		–0.5	-	+0.3V_{DD}	V
V_{IH}	HIGH-level input voltage		0.7V_{DD}	-	5.5	V
I_{OL}	LOW-level output current	V_{OL} = 0.4 V; V_{DD} = 2.3 V	20	28	-	mA
		V_{OL} = 0.4 V; V_{DD} = 5.0 V	30	40	-	mA
I_L	leakage current	V_I = V_{DD} or V_{SS}	–1	-	+1	µA
C_i	input capacitance	V_I = V_{SS}	-	6	10	pF
LED driver outputs						
I_{OL}	LOW-level output current	V_{OL} = 0.5 V; V_{DD} = 2.3 V to 4.5 V [2]	12	25	-	mA
$I_{OL(tot)}$	total LOW-level output current	V_{OL} = 0.5 V; V_{DD} = 4.5 V [2]	-	-	400	mA
I_{OH}	HIGH-level output current	open-drain; V_{OH} = V_{DD}	–10	-	+10	µA
V_{OH}	HIGH-level output voltage	I_{OH} = –10 mA; V_{DD} = 2.3 V	1.6	-	-	V
		I_{OH} = –10 mA; V_{DD} = 3.0 V	2.3	-	-	V
		I_{OH} = –10 mA; V_{DD} = 4.5 V	4.0	-	-	V
I_{OZ}	OFF-state output current	3-state; V_{OH} = V_{DD} or V_{SS}	–10	-	+10	µA
C_o	output capacitance		-	5	8	pF

Table 14. Static characteristics *...continued*

V_{DD} = 2.3 V to 5.5 V; V_{SS} = 0 V; T_{amb} = −40 °C to +85 °C; unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
Address inputs; $\overline{OE}$ input; EXTCLK						
V_{IL}	LOW-level input voltage		−0.5	-	+0.3V_{DD}	V
V_{IH}	HIGH-level input voltage		0.7V_{DD}	-	5.5	V
I_{LI}	input leakage current		−1	-	+1	μA
C_i	input capacitance		-	3	5	pF

[1] V_{DD} must be lowered to 0.2 V in order to reset part.

[2] Each bit must be limited to a maximum of 25 mA and the total package limited to 400 mA due to internal busing limits.

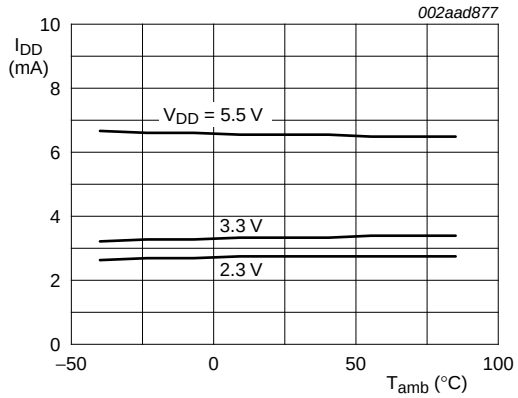


Fig 28. I_{DD} typical values with OSC on and f_{SCL} = 1 MHz versus temperature

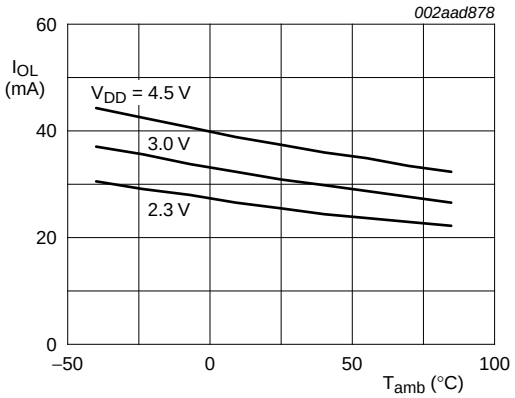


Fig 29. I_{OL} typical drive (LEDn outputs) versus temperature

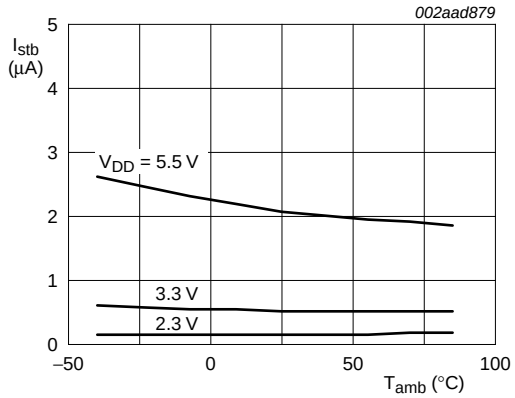


Fig 30. Standby supply current versus temperature

13. Dynamic characteristics

Table 15. Dynamic characteristics

Symbol	Parameter	Conditions	Standard-mode I²C-bus		Fast-mode I²C-bus		Fast-mode Plus I²C-bus		Unit
			Min	Max	Min	Max	Min	Max	
f_{SCL}	SCL clock frequency	[1]	0	100	0	400	0	1000	kHz
f_{EXTCLK}	frequency on pin EXTCLK		DC	50	DC	50	DC	50	MHz
t_{BUF}	bus free time between a STOP and START condition		4.7	-	1.3	-	0.5	-	µs
$t_{HD;STA}$	hold time (repeated) START condition		4.0	-	0.6	-	0.26	-	µs
$t_{SU;STA}$	set-up time for a repeated START condition		4.7	-	0.6	-	0.26	-	µs
$t_{SU;STO}$	set-up time for STOP condition		4.0	-	0.6	-	0.26	-	µs
$t_{HD;DAT}$	data hold time		0	-	0	-	0	-	ns
$t_{VD;ACK}$	data valid acknowledge time	[2]	0.3	3.45	0.1	0.9	0.05	0.45	µs
$t_{VD;DAT}$	data valid time	[3]	0.3	3.45	0.1	0.9	0.05	0.45	µs
$t_{SU;DAT}$	data set-up time		250	-	100	-	50	-	ns
t_{LOW}	LOW period of the SCL clock		4.7	-	1.3	-	0.5	-	µs
t_{HIGH}	HIGH period of the SCL clock		4.0	-	0.6	-	0.26	-	µs
t_f	fall time of both SDA and SCL signals	[4][5]	-	300	$20 + 0.1C_b$ [6]	300	-	120	ns
t_r	rise time of both SDA and SCL signals		-	1000	$20 + 0.1C_b$ [6]	300	-	120	ns
t_{SP}	pulse width of spikes that must be suppressed by the input filter	[7]	-	50	-	50	-	50	ns
t_{PLZ}	LOW to OFF-state propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 10 or 11 in MODE2 register	-	40	-	40	-	40	ns
t_{PZL}	OFF-state to LOW propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 10 or 11 in MODE2 register	-	60	-	60	-	60	ns
t_{PHZ}	HIGH to OFF-state propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 10 or 11 in MODE2 register	-	60	-	60	-	60	ns

Table 15. Dynamic characteristics *…continued*

Symbol	Parameter	Conditions	Standard-mode I²C-bus		Fast-mode I²C-bus		Fast-mode Plus I²C-bus		Unit
			Min	Max	Min	Max	Min	Max	
t_{PZH}	OFF-state to HIGH propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 10 or 11 in MODE2 register	-	40	-	40	-	40	ns
t_{PLH}	LOW to HIGH propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 01 in MODE2 register	-	40	-	40	-	40	ns
t_{PHL}	HIGH to LOW propagation delay	$\overline{OE}$ to LEDn; OUTNE[1:0] = 00 in MODE2 register	-	60	-	60	-	60	ns

[1] Minimum SCL clock frequency is limited by the bus time-out feature, which resets the serial bus interface if either SDA or SCL is held LOW for a minimum of 25 ms. Disable bus time-out feature for DC operation.

[2] $t_{VD;ACK}$ = time for Acknowledgement signal from SCL LOW to SDA (out) LOW.

[3] $t_{VD;DAT}$ = minimum time for SDA data out to be valid following SCL LOW.

[4] A master device must internally provide a hold time of at least 300 ns for the SDA signal (refer to the V_{IL} of the SCL signal) in order to bridge the undefined region of SCL's falling edge.

[5] The maximum t_f for the SDA and SCL bus lines is specified at 300 ns. The maximum fall time (t_f) for the SDA output stage is specified at 250 ns. This allows series protection resistors to be connected between the SDA and the SCL pins and the SDA/SCL bus lines without exceeding the maximum specified t_f.

[6] C_b = total capacitance of one bus line in pF.

[7] Input filters on the SDA and SCL inputs suppress noise spikes less than 50 ns.

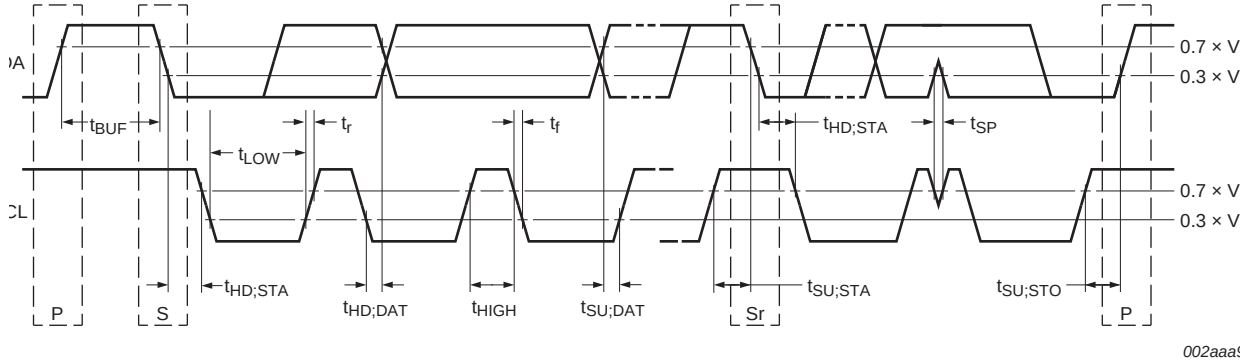
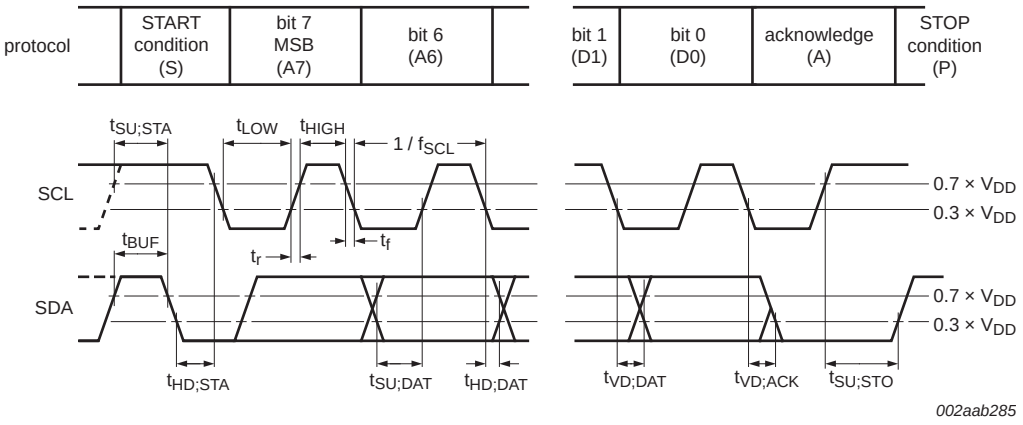


Fig 31. Definition of timing



Rise and fall times refer to V_{IL} and V_{IH}.

Fig 32. I²C-bus timing diagram

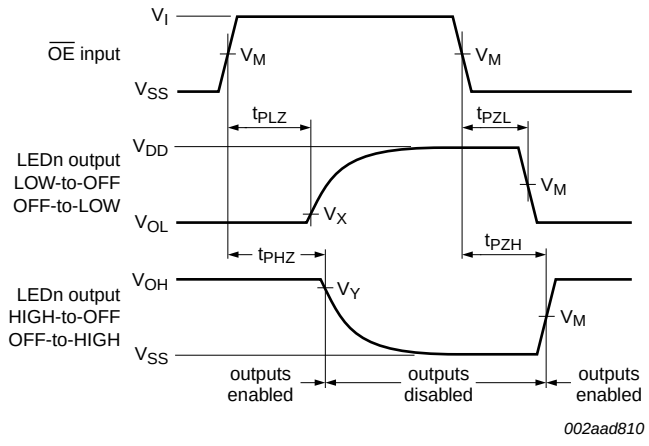
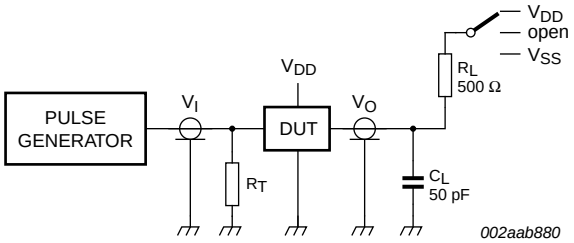


Fig 33. t_{PLZ}, t_{PZL} and t_{PHZ}, t_{PZH} times

14. Test information

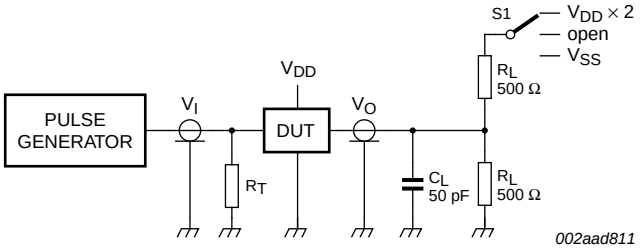


R_L = Load resistor for LEDn.

C_L = Load capacitance includes jig and probe capacitance.

R_T = Termination resistance should be equal to the output impedance Z_o of the pulse generators.

Fig 34. Test circuitry for switching times



R_L = Load resistor for LEDn.

C_L = Load capacitance includes jig and probe capacitance.

R_T = Termination resistance should be equal to the output impedance Z_o of the pulse generators.

Test data are given in Table 16.

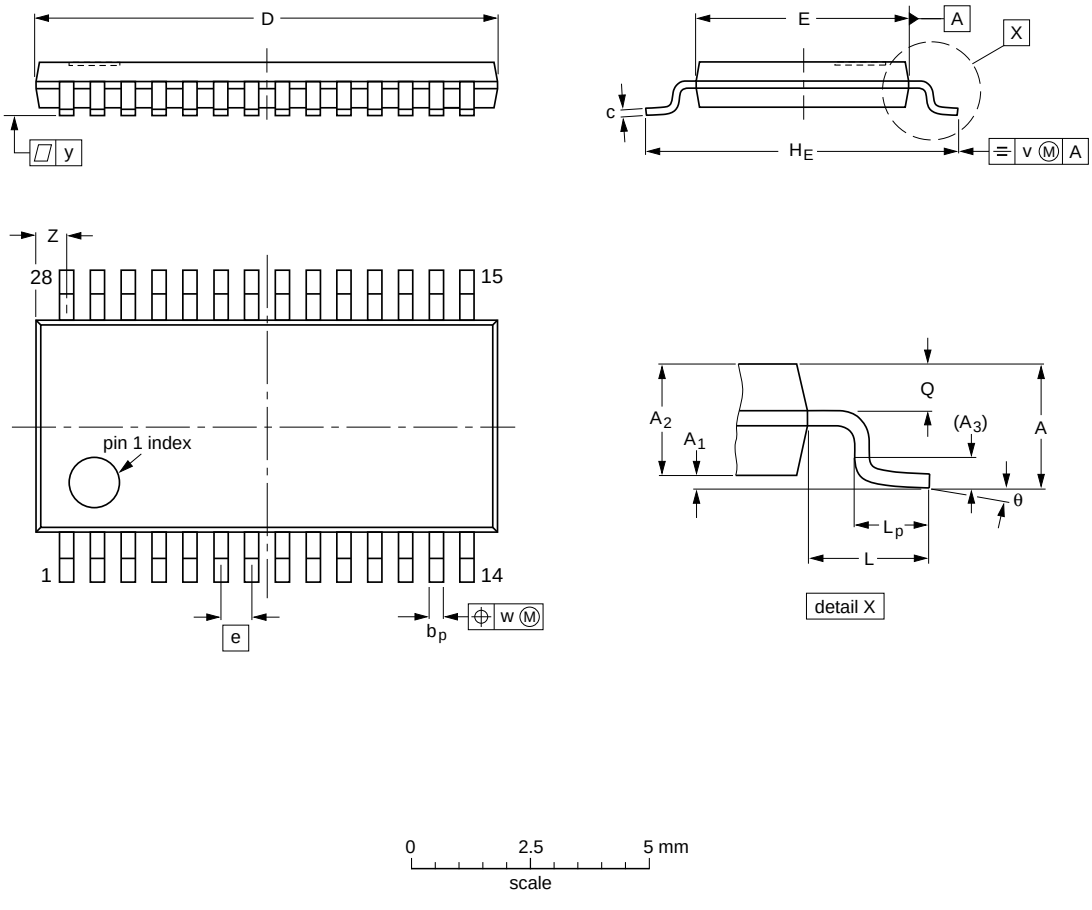
Fig 35. Test circuitry for switching times for enable/disable

Table 16. Test data for enable/disable switching times

Test	Load		Switch
	C_L	R_L	
t_{PD}	50 pF	500 Ω	open
t_{PLZ}, t_{PZL}	50 pF	500 Ω	$V_{DD} \times 2$
t_{PHZ}, t_{PZH}	50 pF	500 Ω	V_{SS}

15. Package outline

TSSOP28: plastic thin shrink small outline package; 28 leads; body width 4.4 mm **SOT361-1**



DIMENSIONS (mm are the original dimensions)

UNIT	A max.	A_1	A_2	A_3	b_p	c	D (1)	E (2)	e	H_E	L	L_p	Q	v	w	y	Z (1)	θ
mm	1.1	0.15 0.05	0.95 0.80	0.25	0.30 0.19	0.2 0.1	9.8 9.6	4.5 4.3	0.65	6.6 6.2	1	0.75 0.50	0.4 0.3	0.2	0.13	0.1	0.8 0.5	8^o 0^o

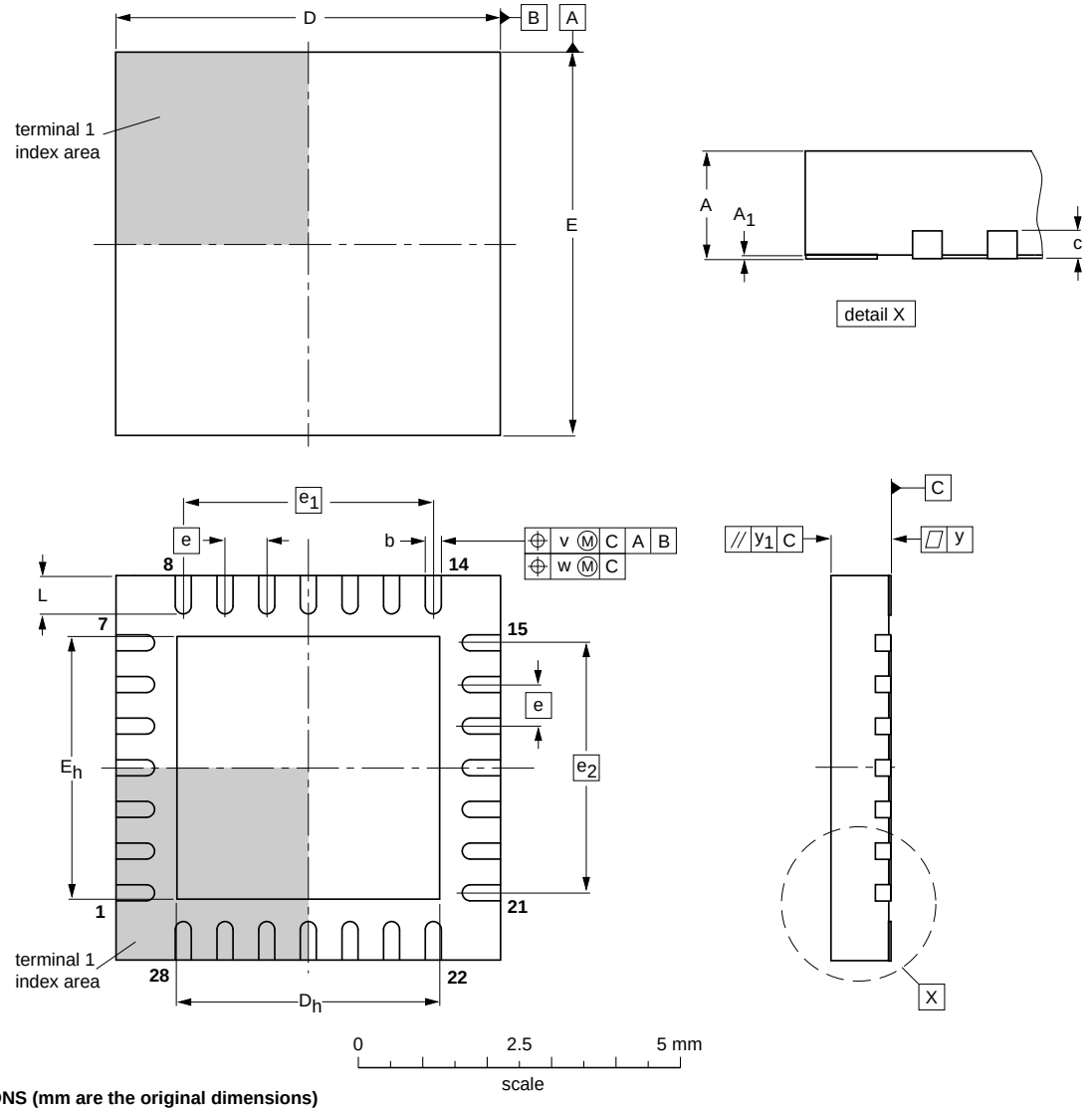
Notes

1. Plastic or metal protrusions of 0.15 mm maximum per side are not included.
2. Plastic interlead protrusions of 0.25 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT361-1		MO-153				~~99-12-27~~ 03-02-19

Fig 36. Package outline SOT361-1 (TSSOP28)

HVQFN28: plastic thermal enhanced very thin quad flat package; no leads; 28 terminals; body 6 x 6 x 0.85 mm **SOT788-1**



DIMENSIONS (mm are the original dimensions)

UNIT	$A^{(1)}$ max.	A_1	b	c	$D^{(1)}$	D_h	$E^{(1)}$	E_h	e	e_1	e_2	L	v	w	y	y_1
mm	1	0.05 0.00	0.35 0.25	0.2	6.1 5.9	4.25 3.95	6.1 5.9	4.25 3.95	0.65	3.9	3.9	0.75 0.50	0.1	0.05	0.05	0.1

Note

1. Plastic or metal protrusions of 0.075 mm maximum per side are not included.

OUTLINE VERSION	REFERENCES				EUROPEAN PROJECTION	ISSUE DATE
	IEC	JEDEC	JEITA			
SOT788-1	- - -	MO-220	- - -			02-10-22

Fig 37. Package outline SOT788-1 (HVQFN28)

16. Handling information

All input and output pins are protected against ElectroStatic Discharge (ESD) under normal handling. When handling ensure that the appropriate precautions are taken as described in *JESD625-A* or equivalent standards.

17. Soldering of SMD packages

This text provides a very brief insight into a complex technology. A more in-depth account of soldering ICs can be found in Application Note *AN10365 "Surface mount reflow soldering description"*.

17.1 Introduction to soldering

Soldering is one of the most common methods through which packages are attached to Printed Circuit Boards (PCBs), to form electrical circuits. The soldered joint provides both the mechanical and the electrical connection. There is no single soldering method that is ideal for all IC packages. Wave soldering is often preferred when through-hole and Surface Mount Devices (SMDs) are mixed on one printed wiring board; however, it is not suitable for fine pitch SMDs. Reflow soldering is ideal for the small pitches and high densities that come with increased miniaturization.

17.2 Wave and reflow soldering

Wave soldering is a joining technology in which the joints are made by solder coming from a standing wave of liquid solder. The wave soldering process is suitable for the following:

- Through-hole components
- Leaded or leadless SMDs, which are glued to the surface of the printed circuit board

Not all SMDs can be wave soldered. Packages with solder balls, and some leadless packages which have solder lands underneath the body, cannot be wave soldered. Also, leaded SMDs with leads having a pitch smaller than ~0.6 mm cannot be wave soldered, due to an increased probability of bridging.

The reflow soldering process involves applying solder paste to a board, followed by component placement and exposure to a temperature profile. Leaded packages, packages with solder balls, and leadless packages are all reflow solderable.

Key characteristics in both wave and reflow soldering are:

- Board specifications, including the board finish, solder masks and vias
- Package footprints, including solder thieves and orientation
- The moisture sensitivity level of the packages
- Package placement
- Inspection and repair
- Lead-free soldering versus SnPb soldering

17.3 Wave soldering

Key characteristics in wave soldering are:

- Process issues, such as application of adhesive and flux, clinching of leads, board transport, the solder wave parameters, and the time during which components are exposed to the wave
- Solder bath specifications, including temperature and impurities

17.4 Reflow soldering

Key characteristics in reflow soldering are:

- Lead-free versus SnPb soldering; note that a lead-free reflow process usually leads to higher minimum peak temperatures (see Figure 38) than a SnPb process, thus reducing the process window
- Solder paste printing issues including smearing, release, and adjusting the process window for a mix of large and small components on one board
- Reflow temperature profile; this profile includes preheat, reflow (in which the board is heated to the peak temperature) and cooling down. It is imperative that the peak temperature is high enough for the solder to make reliable solder joints (a solder paste characteristic). In addition, the peak temperature must be low enough that the packages and/or boards are not damaged. The peak temperature of the package depends on package thickness and volume and is classified in accordance with Table 17 and 18

Table 17. SnPb eutectic process (from J-STD-020D)

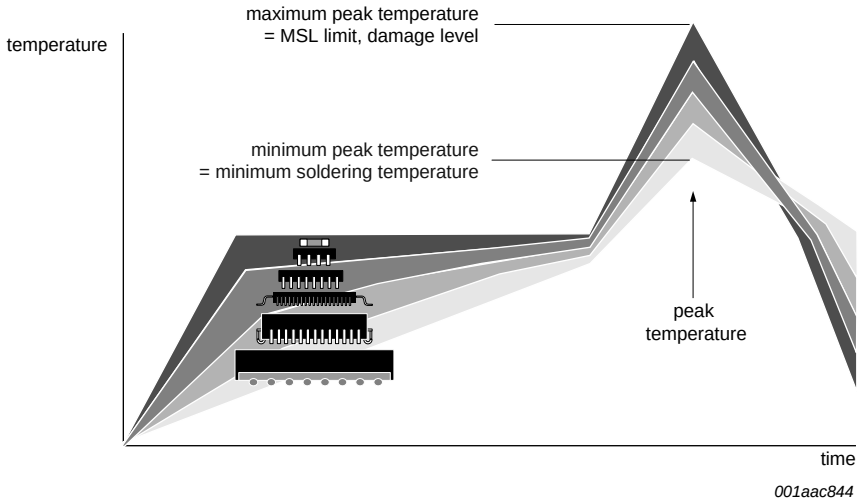
Package thickness (mm)	Package reflow temperature (°C)	
	Volume (mm³)	
	< 350	≥ 350
< 2.5	235	220
≥ 2.5	220	220

Table 18. Lead-free process (from J-STD-020D)

Package thickness (mm)	Package reflow temperature (°C)		
	Volume (mm³)		
	< 350	350 to 2000	> 2000
< 1.6	260	260	260
1.6 to 2.5	260	250	245
> 2.5	250	245	245

Moisture sensitivity precautions, as indicated on the packing, must be respected at all times.

Studies have shown that small packages reach higher temperatures during reflow soldering, see Figure 38.



MSL: Moisture Sensitivity Level

Fig 38. Temperature profiles for large and small components

For further information on temperature profiles, refer to Application Note *AN10365 "Surface mount reflow soldering description"*.

18. Abbreviations

Table 19. Abbreviations

Acronym	Description
CDM	Charged-Device Model
DUT	Device Under Test
EMI	ElectroMagnetic Interference
ESD	ElectroStatic Discharge
HBM	Human Body Model
I2C-bus	Inter-Integrated Circuit bus
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LSB	Least Significant Bit
MM	Machine Model
MSB	Most Significant Bit
NMOS	Negative-channel Metal-Oxide Semiconductor
PCB	Printed-Circuit Board
PMOS	Positive-channel Metal-Oxide Semiconductor
POR	Power-On Reset
PWM	Pulse Width Modulation; Pulse Width Modulator
RGB	Red/Green/Blue
RGBA	Red/Green/Blue/Amber
SMBus	System Management Bus

19. Revision history

Table 20. Revision history

Document ID	Release date	Data sheet status	Change notice	Supersedes
PCA9685 v.4	20150416	Product data sheet	-	PCA9685 v.3
Modifications:	• Changed programmable frequency to “24 Hz to 1526 Hz” throughout • Minor edits to text and figures to provide clarity regarding cycle count throughout			
PCA9685 v.3	20100902	Product data sheet	-	PCA9685 v.2
PCA9685 v.2	20090716	Product data sheet	-	PCA9685 v.1
PCA9685 v.1	20080724	Product data sheet	-	-

20. Legal information

20.1 Data sheet status

Document status[1][2]	Product status[3]	Definition
Objective [short] data sheet	Development	This document contains data from the objective specification for product development.
Preliminary [short] data sheet	Qualification	This document contains data from the preliminary specification.
Product [short] data sheet	Production	This document contains the product specification.

[1] Please consult the most recently issued document before initiating or completing a design.

[2] The term 'short data sheet' is explained in section "Definitions".

[3] The product status of device(s) described in this document may have changed since this document was published and may differ in case of multiple devices. The latest product status information is available on the Internet at URL http://www.nxp.com.

20.2 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

Short data sheet — A short data sheet is an extract from a full data sheet with the same product type number(s) and title. A short data sheet is intended for quick reference only and should not be relied upon to contain detailed and full information. For detailed and full information see the relevant full data sheet, which is available on request via the local NXP Semiconductors sales office. In case of any inconsistency or conflict with the short data sheet, the full data sheet shall prevail.

Product specification — The information and data provided in a Product data sheet shall define the specification of the product as agreed between NXP Semiconductors and its customer, unless NXP Semiconductors and customer have explicitly agreed otherwise in writing. In no event however, shall an agreement be valid in which the NXP Semiconductors product is deemed to offer functions and qualities beyond those described in the Product data sheet.

20.3 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Limiting values — Stress above one or more limiting values (as defined in the Absolute Maximum Ratings System of IEC 60134) will cause permanent damage to the device. Limiting values are stress ratings only and (proper) operation of the device at these or any other conditions above those given in the Recommended operating conditions section (if present) or the Characteristics sections of this document is not warranted. Constant or repeated exposure to limiting values will permanently and irreversibly affect the quality and reliability of the device.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at http://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

No offer to sell or license — Nothing in this document may be interpreted or construed as an offer to sell products that is open for acceptance or the grant, conveyance or implication of any license under any copyrights, patents or other industrial or intellectual property rights.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Non-automotive qualified products — Unless this data sheet expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

20.4 Trademarks

Notice: All referenced brands, product names, service names and trademarks are the property of their respective owners.

21. Contact information

For more information, please visit: **http://www.nxp.com**

For sales office addresses, please send an email to: **salesaddresses@nxp.com**

22. Contents

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 16 April 2015
Document identifier: PCA9685