PROYECTO FIN DE CARRERA

MEJORAS EN EL CONTROL Y SISTEMA DE COMUNICACIONES DEL ROBOT BALLBOT-ULL

Autor: Francisco Raúl Machín Castilla
Director: Santiago Torres Álvarez

ÍNDICE DE ILUSTRACIONES

SECCIÓN DE INGENIERÍA INFORMÁTICA

Capítulo 1

Aspectos generales del proyecto

Titulación: Ingeniería en Informática

Alumno: Francisco Raúl Machín Castilla

Tutor: Santiago Torres Álvarez

Julio 2016

Contenido

1.1. Aspectos generales del proyecto

1.1.1. Antecedentes

El proyecto se redacta como propuesta inicial del tutor Dr. Santiago Torres Álvarez para la presentación en la asignatura "Proyecto Fin de Carrera", perteneciente a la asignatura anual del segundo curso de Ingeniería Informática, impartida en la Universidad de La Laguna.

Este proyecto forma parte un proyecto mucho más extenso, que supone la construcción de un robot móvil y autónomo capaz de recibir información del entorno que le rodea y de ser capaz, tras recibir dicha información, de recoger pelotas inmóviles de una pista de tenis.

La implementación electrónica del sistema de alimentación y la fabricación del prototipo , comprenden otros proyectos diferentes.

El alumno Francisco Raúl Machín Castilla (con DNI: 43822064-A) solicita la ejecución de este proyecto bajo la dirección de Don Santiago Torres Álvarez, profesor y Doctor contratado del Departamento de Ingeniería Informática y de Sistemas.

1.1.2. Objeto del proyecto

El primer objeto del proyecto es la culminación en la obtención del título en Ingeniería Informática y en segundo lugar, mejorar el diseño y movimiento del robot móvil autónomo capaz de recoger pelotas de tenis que se encuentran inmóviles en una cancha de tenis, tanto en hardware como en software.

Podemos destacar las siguientes grandes partes en este proyecto:

1) Mejoras en el diseño del robot. Todas ellas implementadas en el diseño de su electrónica. Simplicidad en la circuitería y liberación del peso al que se ve sometido el robot durante su desplazamiento para detectar las pelotas de tenis, acercarse hasta ellas y recogerlas una a una, almacenarlas en su interior y depositarlas en un lugar determinado indicado.

2) Mejoras de Velocidad. Utilizando únicamente un PC a bordo y mediante la comunicación Wifi, se incrementa la velocidad de recogida de las pelotas de tenis hasta de al menos, un 75% o más, durante todo el proceso.

3) Simplicidad y mayor eficiencia en cuanto a toda la programación necesaria para realizar el proceso de recogida de pelotas. Mayor sencillez en código, mejor entendimiento del mismo, menos coste computacional y menor tiempo de ejecución del código.

1.1.3. Condiciones de partida

Se dispondrá de una estructura física, motores y ruedas del prototipo ya diseñado y construido en fases previas del proyecto.

Así mismo también se dispondrá de unos componentes electrónicos adecuados para la implementación del proyecto y de un Pc a bordo encargado de gobernar y dirigir dichos componentes.

Por tanto, y como parte del presente proyecto, será necesario realizar tareas de mejora en su estructura electrónica e implementar los algoritmos necesarios sobre dicha plataforma para que el robot pueda llevar a cabo sus tareas de forma autónoma.

1.1.4. Peticionario

El peticionario de este proyecto es la Escuela Superior de Ingeniería y Tecnología perteneciente a la Universidad de La Laguna, ubicada en Camino San Francisco de Paula, s/n C.P. 38271. Campus de Anchieta, La Laguna.

1.1.5. Ámbito del proyecto

Los conceptos implicados durante el desarrollo de este proyecto incluyen, entre otras, a las asignaturas de Robótica, Programación Concurrente, Fundamentos Matemáticos y Físicos de la Ingeniería, Fundamentos Electrónicos, Diseños de Sistemas Operativos, Procesamiento de Señales y Control Inteligente.

Entre dichos conceptos destaca:

- Técnicas de comunicación y transmisión.
- Tratamiento de señales digitales y analógicas.
- Memoria compartida y procesamiento paralelo.
- Técnicas de control.
- Aplicación de la cinemática directa y/o inversa.
- Aplicación de lenguajes de programación de alto nivel.

1.1.6. Emplazamiento

El lugar en el que se desarrolla el proyecto, a parte de emplazamiento

particular, es en el laboratorio del Departamento de Ingeniería Informática y de Sistemas, sito en la Facultad de Ciencias Sección de Física de la Universidad de La Laguna.

1.1.7. Recursos utilizados

Se ha utilizado software libre y específico para la programación del ordenador de placa simple (SBC) de bajo coste Raspberry Pi en el sistema operativo Linux.

Para el diseño de la electrónica necesaria e implementación en la protoboard se han utilizado componentes electrónicos para el circuito correspondiente en la lectura de encoders.

SECCIÓN DE INGENIERÍA INFORMÁTICA

Capítulo 2

Pc-OnBoard: Raspberry Pi

Titulación: Ingeniería en Informática

Alumno: Francisco Raúl Machín Castilla

Tutor: Santiago Torres Álvarez

Julio 2016

Contenido

2.1. Robótica móvil, descripción del prototipo de robot móvil

2.1.1 Robot móvil

Un robot móvil podría definirse como una máquina automática que es capaz de trasladarse en cualquier ambiente dado.

Los robots móviles tienen la capacidad de moverse en un determinado entorno y no se encuentran fijados en una ubicación en concreto. Según el medio donde desarrollan su movimiento podríamos clasificarlos como terrestres, acuáticos, aéreos e híbridos.

Podríamos establecer la siguiente analogía entre un robot y una persona:

* El computador corresponde al cerebro

* Los sensores y transductores corresponden a los 5 sentidos

* Los mecanismos ejecutores del robot a los músculos

* El mecanismo estructural del robot al esqueleto

* La fuente de alimentación al metabolismo

2.1.2. Descripción del prototipo

El robot móvil que describe este documento es un robot autónomo y terrestre. Sus unidades de entrada son las siguientes:

- Una webcam para recoger información del entorno que lo rodea.

- Encoders encargados de aportar la información necesaria para conocer el desplazamiento y giro de los motores.

- Electrónica necesaria para el conocimiento del estado de la batería y del resto de componentes externos.

Entre las unidades de salida destaca:

- Los motores encargados del movimiento.

- Los servos encargados de recoger y soltar las pelotas de tenis.

- Indicadores de luz que informan del nivel de batería y de la información recogida a través de los encoders.

La unidad de procesamiento está dividida en dos módulos:

- Un PC, encargado de mantener comunicación en todo momento con el robot.

- Un PC a bordo, encargado de realizar la detección de las pelotas y de enviar las instrucciones pertinentes y señales determinadas a los servos y motores para su movimiento. Se precisa de una comunicación entre el PC central para operaciones de monitorización de los distintos elementos del robot.

El PC remoto con el que se comunica el robot, lleva a cabo de tareas de posicionamiento global del robot en la pista y las tareas de control para la ejecución de las trayectorias de recogida de pelotas. Una vez este PC emplace al robot en las cercanías de la zona de recogida, será el propio robot quien, de forma autónoma, lleve a cabo el acercamiento preciso a la pelota y su recogida.

2.2 Elección de Raspberry Pi como dispositivo de centralización

2.2.1. Descripción

Como ya hemos descrito anteriormente, Raspberry Pi es un ordenador de placa reducida o simple (SBC) de bajo coste que ha sido desarrollada en el Reino Unido con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas aunque no empezó su comercialización hasta el año 2012. El concepto es el de un ordenador sin todos los accesorios que se pueden eliminar sin que afecte a su funcionamiento básico, guarda en su interior un extraordinario poder de cómputo.

Aunque no se indica si es hardware libre o con derechos de marca, en su sección de respuestas frecuentes explican que disponen de contratos de distribución y venta con dos empresas pero que al mismo tiempo, cualquiera puede revender o redistribuir tarjetas Raspberry Pi, por lo que se entiende que es un producto con propiedad registrada pero de uso libre. Todo lo contrario que su software, que sí lo es libre en su totalidad, siendo su sistema operativo oficial una versión adaptada de las distribuciones Debian (Raspbian).

El diseño incluye:

- Un “System-On-A-Chip” Broadcom BCM2835, que contiene un procesador central (CPU) ARM1176JZF-S a 700 MHz (el firmware incluye unos modos “Turbo” para que el usuario pueda hacerle overclock de hasta 1 GHz sin perder la garantía).

- Un procesador gráfico (GPU) VideoCore IV.

- 512 MB de memoria RAM aunque originalmente al ser lanzado eran 256 MB.

- Incluye 2 buses USB 2.0.

- Una salida analógica de audio estéreo por Jack de 3.5mm.

- Salida digital de vídeo + audio HDMI.

- Salida analógica de video RCA.

- Pines de entrada y salida de propósito general.

- Conector de alimentación microUSB.

- El diseño no incluye un disco duro o una unidad de estado sólido, ya que usa una tarjeta SD para el almacenamiento permanente; tampoco incluye fuente de alimentación o carcasa.

La fundación da soporte para las descargas de las distribuciones para arquitectura ARM, “Raspbian” (derivada de “Debian”), “RISC OS 5”, “Arch Linux ARM” (derivado de “Arch Linux”) y “Pidora” (derivado de “Fedora”); y promueve principalmente el aprendizaje del lenguaje de programación Python que es el lenguaje utilizado en el desarrollo de este proyecto, y otros lenguajes como Tiny BASIC, C y Perl.

Para conocer más detalles de la historia de Raspberry Pi ver los anexos en la parte final de esta memoria.

2.2.2. Hardware

Actualmente existen al menos 5 modelos diferentes de Raspberry Pi que podemos adquirir a través de su web: Raspberry Pi 1 Model A+, Raspberry Pi 1 Model B+, Raspberry Pi Zero, Raspberry Pi 2 Model B y Raspberry Pi 3 Model B.

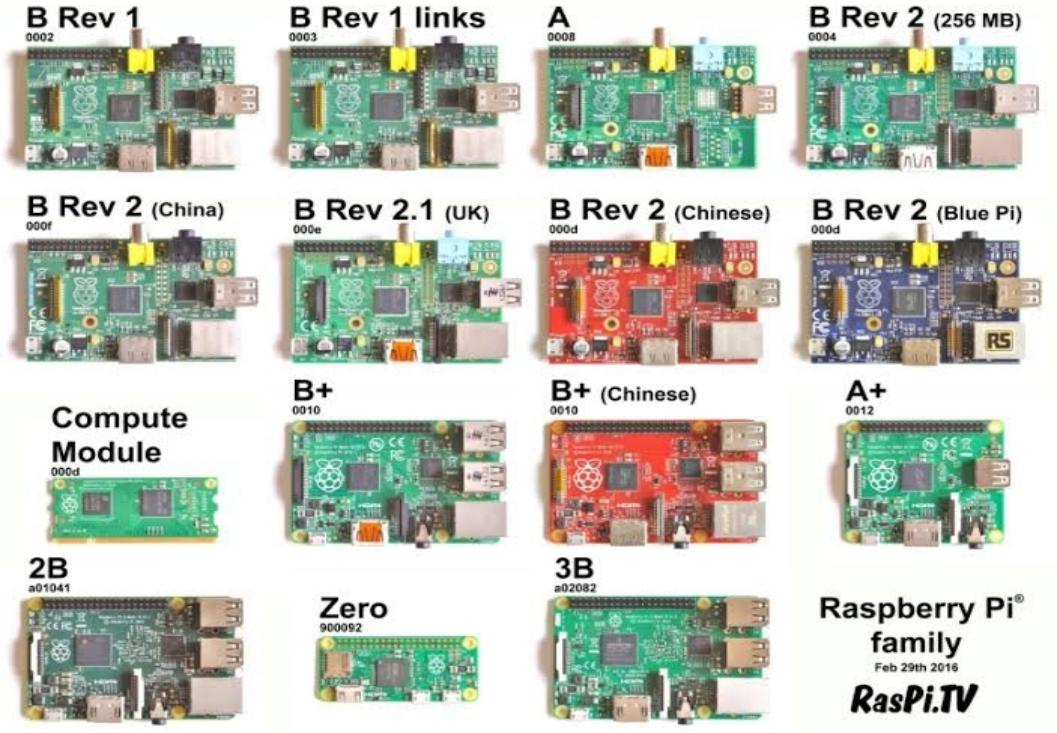


Ilustración 1: Tipos de Placas Raspberry Pi a 29 Feb 2016

La Raspberry Pi en la cual está basado este proyecto es el modelo B Rev2.

El hardware que viene implementado en la placa presenta las siguientes características:

Microprocesador

El procesador en el interior de la Raspberry Pi es un procesador multimedia Broadcom BCM2835 system-on-chip (SoC). Esto quiere decir que la mayor parte de lo componentes del sistema, incluidos CPU y GPU, junto con el audio y el hardware necesario para las comunicaciones, se encuentran integrados dentro de aquel único componente ubicado justo debajo del chip de la memoria de 512 MB en el centro de la placa. A parte del diseño del SoC, la arquitectura de conjunto de instrucciones es ARM, esto es precisamente, lo que lo diferencia de un procesador de un PC o de un ordenador portátil.

La combinación de la arquitectura RISC (Simple Reduced Instruction Set) y su bajo consumo energético lo convierten en la opción perfecta frente a los chips de computadoras de escritorio que demandan altos consumos y arquitecturas más complejas CISC (complex instruction set).

Es por tanto, el microprocesador el secreto que explica cómo la Raspberry Pi es capaz de funcionar con tan sólo una fuente de alimentación de 5V 1A suministrada por el puerto microUSB que se describirá más adelante. También el BCM2835 es la razón por la cuál no hay ningún disipador térmico sobre el dispositivo: el bajo consumo de energía del chip se traduce directamente en muy poco calor residual, incluso durante las tareas más complejas de procesamiento a las que hace frente.

Todo esto significa que la Raspberry Pi es incompatible con el software de los PC tradicionales. La mayoría del software para ordenadores de escritorio y portátiles se construyen teniendo en cuenta la arquitectura de conjunto de instrucciones x86, presente en los procesadores como AMD, Intel y VIA.

El BCM2835 utiliza una generación del diseño del procesador ARM conocida como ARM11, que a su vez, está diseñada en torno a una versión de la arquitectura de conjunto de instrucciones conocida como ARMv6. Vale la pena recordar que ARMv6 es una arquitectura ligera y potente pero que tiene un rival importante en la arquitectura más avanzada, el ARMv7 utilizada por la familia de procesadores ARM Cortex. El software desarrollado para la ARMv7 es, por desgracia, incompatible con el BCM2835 de la Raspberry Pi, aunque los desarrolladores generalmente pueden convertir el sofware para adecuarlo.

Hay un montón de software disponible para el conjunto de instrucciones ARMv6 y, en tanto que la popularidad de la Raspberry Pi sigue aumentando, el software disponible va a seguir creciendo.

CPU

La CPU contiene un ARM1176JZFS, con unidad de coma flotante, que funciona a 700 MHz y que es capaz de soportar un "overclock" de hasta un 1GHz en modo "TURBO" que hace que el SoC de más rendimiento sin reducir el tiempo de vida de la placa y sin perder la garantía. La CPU está basada en la versión 6 de la arquitectura ARM, la cual no es soportada por una gran cantidad de distribuciones de Linux, incluyendo Ubuntu.

GPU

La GPU utilizada es una Dual Core VideoCore IV Multimedia Co-Processor que es capaz de mover contenidos con calidad BlueRay, usando H.264 hasta 40MBits/segundo. Dispone de un núcleo 3D con soporte para las librerías OpenGL ES2.0 y OpenVG. Es capaz de decodificar hasta 1080p (no entrelazada) y 30 fotogramas por segundo.

RAM

La memoria RAM es de 512MB de SDRAM en un único módulo, el cual, funciona a 400MHz en su modo normal alcanzando los 600MHz en su versión "TURBO".

ALMACENAMIENTO

La Raspberry Pi no tiene un disco duro, para ello dispone de un lector/ranura para memorias SD, un sistema de almacenamiento en estado sólido. El arranque del sistema se realiza desde la propia tarjeta del sistema con lo que, al albergar el espacio todo el sistema operativo, es necesario que la tarjeta sea de al menos 2GB de capacidad para almacenar todos los archivos requeridos. Como es obvio, a medida que las versiones de los sistemas operativos avanzan, empezarán a ocupar más espacio en la tarjeta. Por tanto, actualmente, se recomienda un espacio mínimo de 4GB y muchas veces, dependiendo de la cantidad de archivos generados, lo recomendable ya está rondando los 8GB de espacio en la tarjeta. Incluso, si queremos asegurarnos de una óptima rapidez en el sistema operativo de nuestra Raspberry Pi, podemos utilizar diferentes tipos de tarjetas que se utilizan para fotografía en el mercado. Las tarjetas SD miden su velocidad de escritura de datos en base a la velocidad máxima, las de 40x escriben a 6MB por segundo, las de 80x a 12MB, las de 100x a 15MB, etc. Sin embargo, si queremos garantizar una calidad mínima, nos interesa pasar a una tarjeta SDHC, cuya clase mide la velocidad mínima de escritura. Así, una clase 2 escribe a 2MB por segundo, una clase 6 a 6MB por segundo y por último, una clase 10 a 10MB por segundo. Raspberry Pi acepta tarjetas SDHC.

Están disponibles tarjetas SD con el sistema operativo precargado en la tienda oficial, si no tuviéramos posibilidad de adquirirla, será necesario instalar (flashear) un sistema operativo en la tarjeta antes de poder trabajar con ella. También existe la posibilidad, tras el arranque inicial de la SD se puede trabajar con almacenamiento de algún dispositivo de disco por USB.

SALIDAS DE VIDEO

Para la salida de video la Raspberry Pi posee un conector RCA o Video Compuesto (PAL y NTSC), un conector HDMI (rev 1.3 y rev 1.4) y una interfaz DSI para paneles LCD.

El vídeo compuesto está diseñado para poder conectar a la Raspberry a los antiguos dispositivos de pantalla. El conector crea una señal compuesta de colores que se encuentra dentro de una imagen (rojo, verde y azul) y lo envía por un solo cable hacia la pantalla, suele tratarse de las antiguas televisiones de rayos catódicos (CRT). La señal de imagen es una señal de menor calidad y además son más propensas a las interferencias, falta de claridad de imagen y tienen una resolución bastante limitada.

Para mejorar la calidad de imagen la Raspberry cuenta con un conector HDMI (High Definition Multimedia Interface). A diferencia de la conexión del vídeo compuesto, la señal por HDMI proporciona una conexión digital de alta velocidad para mostrar imágenes de píxeles perfectos tanto en monitores de PC como en televisiores de alta definición. La resolución de la imagen que puede alcanzar esta salida digital ronda los 1920x1080 Full HD. Con estos niveles de resolución, el detalle sobre la pantalla es mucho mayor.

La última salida que dispone Raspberry es conocida como Display Serial Interface (DSI) y es utilizado en los monitores de pantalla plana de las tablets y de los smartphones.

SALIDAS DE AUDIO

Además del propio HDMI que, cuando está configurado correctamente, transporta ambas señales, tanto la de vídeo como la de audio, Raspberry Pi posee un conector de audio Jack de 3.5mm. Normalmente cuando el display no tiene entrada HDMI, se utilizaría la salida de audio Jack si quisiéramos sacar el sonido.

BUS USB

El modelo B de Raspberry Pi posee 2 puertos USB 2.0 (vía hub USB integrado). Es a través de estos puertos donde conectaríamos dispositivos usb tales como discos duros, impresoras, adaptadores de red usb, etc.

TARJETA DE RED

La Raspberry posee un conector RJ-45 conectado a un integrado lan9512-jzx de SMSC que nos proporciona una conectividad de hasta 10/100 Mbps. Con esto, es posible conectar directamente la Raspberry a un PC sin tener que pasar por un router. Dicha conexión no hace falta hacerla con un cable cruzado ya que el

conector de red incluye una característica conocida como auto-MDI lo que le permite reconfigurarse automáticamente.

Recientemente se ha lanzado la Rapsberry Pi 3 con adaptador wifi integrado por lo que la característica integrada para gestionar redes inalámbricas en los modelos anteriores es imposible si no añadimos un adaptador USB para red inalámbrica (incluyendo aquellas redes que cumplen el estándar 802.11n)

PINES DE ENTRADA Y SALIDA DE PROPÓSITO GENERAL

Para las entradas y salida sin un propósito específico, la Raspberry Pi posee un conector de 26 pines conocido como GPIO y cuyo comportamiento de esos pines (tanto si son pines de entrada o de salida) se pueden programar por el usuario en tiempo de ejecución. Algunos modelos de Raspberry incorporan un zócalo con 8 pines más que también pueden programarse. En el caso donde se desarrolla este proyecto, dichos pines están incorporados.

ENERGÍA Y ALIMENTACIÓN

Raspberry Pi carece de botón de encendido y apagado, aunque es verdad que viene un zócalo en la placa por si queremos incorporarle un botón de apagado soldándolo, la energía le llega mediante un conector microUSB estándar de 5 Voltios. El consumo de la placa es de 700mA (3.5 Watios)

Por tanto, muchos cargadores diseñados para los smartphones funcionarán con la Raspberry pero hay que tener cuidado ya que algunos sólo suministran 500mA y la placa consume más energía que la mayoría de los dispositivos microUSB y sigue requiriendo de almenos, los citados 700mA para poder funcionar correctamente.

2.2.3. Software

Otra diferencia importante entre Raspberry Pi y el PC de escritorio convencional, a parte de su tamaño y su coste, es el sistema operativo, es decir, aquel programa o conjunto de programas de un sistema informático que gestiona los recursos de hardware y provee servicios a los programas de aplicación de software, ejecutándose en modo privilegiado respecto a los restantes (aunque puede que parte de él se ejecute en espacio de usuario).

La mayoría de los PCs y portátiles disponibles hoy en día funcionan con alguno de estos dos sistemas operativos: Microsoft Windows o Apple OS X. Ambas plataformas son de código cerrado, creados en un ambiente reservado utilizando técnicas patentadas. Estos sistemas operativos son conocidos como de código cerrado por la naturaleza de su código fuente, es decir, la receta en lenguaje de computadora que le dice al sistema que hacer. En el software de código

cerrado, esta receta es mantenida como un secreto muy bien guardado. Los usuarios pueden obtener el software terminado, pero nunca ver cómo está hecho.

Sin embargo, si queremos que Raspberry Pi sirva para aprender en las escuelas, necesitamos que esté diseñada para ejecutarse en un sistema operativo de código abierto. Esto es, GNU/Linux. Esto quiere deicr que es posible descargar el código fuente del sistema operativo por completo y hacer los cambios que se desee. Nada es ocultado, todos los cambios hechos están a la vista del público. Este espíritu de desarrollo de código abierto ha permitido a Linux rápidamente ser modificado para poder ejecutarse sobre la Raspberry Pi, un proceso que es conocido como “portabilidad”.

Varias versiones de Linux (conocidas como distribuciones) han sido portadas al chip BCM2835 de la Raspberry Pi, incluyendo Debian, Fedora Remix y Arch Linux. Las distintas distribuciones atienden diferentes necesidades, pero todas ellas tienen algo en común: son de código abierto. Además, por lo general, todas son compatibles entre sí: el software escrito en un sistema Debian funcionará perfectamente bien en uno con Arch Linux y viceversa.

Igual que con la diferencia entre la arquitectura ARM y la x86, hay un punto clave que hace la diferencia práctica entre Windows, OS X y Linux: el software escrito para Windows u OS X no funciona en Linux. Afortunadamente, hay un montón de alternativas compatibles para la gran mayoría de los productos de software comunes y lo mejor, casi todos son de libre uso y de código abierto como lo es el propio sistema operativo.

Existen una serie de sistemas operativos para Raspberry Pi entre los que podríamos destacar:

- Raspbian
- Android
- Arch Linux
- Debian
- Firefox OS
- Google Chromium OS
- Kali Linux
- Pidora
- Slackware
- RISC OS 52
- FreeBSD

- NetBSD

Algunas distribuciones ligeras multipropósito:

- Moebius
- Squeezed Arm Puppy (Puppy Linux)
- Minibian (Raspbian)

Otras distribuciones ligeras de único propósito:

- Instant Webkiosk (S.O. con solo un navegador)
- IPFire
- Micro Elastix
- OpenELEC (computadora personal de cine en casa)
- Raspbmc (Multimedia)
- Xbian (Multimedia)

2.3. Módulos del proyecto

Este proyecto está dividido en 3 módulos claramente diferenciados que han sido modificados y mejorados:

- Lógica de comportamiento: Encargado de describir e implementar la lógica general del funcionamiento del robot.
- Módulo de comunicación: Encargado de transmitir la información entre el robot y el ordenador principal de pista.
- Módulo de control: Encargado de realizar las acciones de desplazamiento y recogida de pelotas.

SECCIÓN DE INGENIERÍA INFORMÁTICA

Capítulo 3

Módulo de Comunicación

Titulación: Ingeniería en Informática

Alumno: Francisco Raúl Machín Castilla

Tutor: Santiago Torres Álvarez

Julio 2016

Contenido

3 MODULO DE COMUNICACIÓN

3.1. Introducción al módulo de comunicación

Para poder comunicarnos con el robot móvil a distancia necesitamos del sistema de comunicación que nos habilitará para ello. Un sistema de comunicación es cualquier sistema que permite establecer una comunicación a través de él. Al menos en un sistema de comunicación intervienen los siguientes elementos básicos:

- Emisor: La parte del sistema que codifica y emite el mensaje.

- Receptor: todo dispositivo capaz de recibir un mensaje y extraer la información de él.

- Medio de transmisión: el soporte físico por el que se transmite la información.

Para que un sistema de comunicación sea efectivo debe satisfacer tres necesidades esenciales:

- Entrega: El sistema garantiza la llegada de toda la información donde debe y sólo va a ser recibida donde estaba previsto.

- Exactitud: El sistema debe entregar la información con la mayor exactitud posible y sin modificaciones. Los datos que se alteren en la transmisión deben de poder recuperarse a través de códigos detectores y correctores de error u otras técnicas.

- Puntualidad: El sistema debe entregar la información en el intervalo de tiempo previsto para ello. En el caso de transmisión en tiempo real (vídeo u audio) la entrega de datos debe efectuarse sin que se produzcan retrasos significativos.

3.2. Protocolo VNC y configuraciones para la comunicación

VNC (Virtual Network Computing) es un protocolo o método estándar para compartir el escritorio de un ordenador que interconecta ordenadores y sistemas operativos diferentes. Aunque el protocolo no es muy seguro, a pesar de su encriptación, puede ser utilizado a través de otro protocolo llamado SSH o a través de una red virtual haciéndolo mucho más seguro.

Actualmente VNC puede encontrarse en un programa de software libre y se basa en una estructura cliente-servidor el cual le permite tomar el control del ordenador servidor remotamente a través de un ordenador que hace de cliente. Por tanto, hablamos de una aplicación con dos componentes o programas (cliente y servidor) que se instalan en los elementos donde queremos establecer la comunicación siendo sólo requerido el primer programa para conectar con los ordenadores de la Unidad, por lo que el usuario puede omitir la instalación del segundo (el servidor). Es decir, basta sólo tener instalada la aplicación cliente en los equipos que quieran conectarse a otro equipo y precisaremos del programa servidor cuando queramos que algún PC se conecte con nuestro equipo.

VNC no impone restricciones en el sistema operativo del ordenador servidor con respecto al del cliente: es posible compartir la pantalla de una máquina con cualquier sistema operativo que soporte VNC conectándose desde otro ordenador o dispositivo que disponga de un cliente VNC portado.

Un ejemplo de uso de VNC podría darse reflejado en un profesor teniendo que impartir una clase de laboratorio a sus alumnos. El profesor a la vez que realiza las explicaciones puede mostrarlas a sus alumnos compartiendo su pantalla o por ejemplo y más común en la vida real, un técnico que precisa resolver remotamente un problema que informa un usuario inexperto.

Para poder establecer una conexión mediante protocolo VNC con la Rapsberry Pi se deben realizar las siguientes acciones y configuraciones:

1º Obtención del servidor y cliente

Los pasos en orden a seguir para la instalación del servidor en la Raspberry Pi han sido los siguientes:

- Actualización de paquetes.

Abrimos una terminal o consola en Raspberry si estamos en el modo gráfico, si no, estaremos ya sobre la consola y bastará con ejecutar los siguientes comandos:

sudo apt-get update && sudo apt-get upgrade

- Instalación del servidor VNC en Raspberry.

sudo apt-get install tightvncserver

Tightvnc es una aplicación informática de código libre para la administración remota multiplataforma que utiliza el protocolo RFB de VNC para controlar la pantalla de otro equipo de forma remota. En el protocolo Remote Frame Buffer, la aplicación que se ejecuta en el equipo que utiliza el usuario (que contiene la pantalla, el teclado y el puntero) se llama cliente. La aplicación que se ejecuta en el equipo donde se encuentra el frame-buffer (el cual ejecutan el sistema de ventanas y aplicaciones que controla el usuario de manera remota) se llama servidor. Lleva una cantidad razonable de tráfico de red enviar una imagen del frame-buffer, así Remote Frame Buffer funciona mejor sobre enlaces de banda ancha, como redes de área local.

2º Configuración del servidor

- Configuración del Servidor VNC en Raspberry Pi.

A continuación ejecutaremos la siguiente orden desde el terminal de nuestra Raspberry:

tightvncserver

En este momento la aplicación pedirá una password, esta contraseña es la que se tendrá que escribir cada vez que queramos acceder a la Raspberry Pi desde el cliente. He seguido la nomenclatura de utilizar como contraseña la palabra " r*aspberry* ". Tras esto responderemos negativamente a la pregunta:

Would you like to enter a view-only password (y / n) ? ***n***

Esta pregunta nos indica si queremos utilizar una segunda contraseña que pueda ser utilizada para la autenticación en modo de sólo lectura. Esta contraseña restringirá el ratón y el teclado de entrada a los clientes que se autentiquen con ella.

A continuación modificaremos el fichero " *etc/rc.local* " con el objetivo de crear una configuración determinada al terminar el arranque de nuestra Raspberry Pi y no tenerlo que hacer cada vez que iniciamos. Es decir, cualquier comando que coloquemos o script al que llamemos en dicho fichero, será ejecutado al final del arranque, cuando todos los scripts que tenemos en el runlevel o nivel de ejecución correspondiente hayan sido ejecutados.

Inicialmente el archivo etc/rc.local se encuentra así:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

exit 0
```

A continuación añadiremos la siguiente línea:

su -c "/usr/bin/tightvncserver -geometry 1280x800 -depth 16 -pixelformat rgb565" pi

Con lo que conseguiremos que cada vez que iniciemos la Raspberry Pi el servidor de VNC esté operativo con una resolución de pantalla de 1280 x 800 pixeles para el usuario " *pi* "

El fichero una vez modificado queda de la siguiente forma:

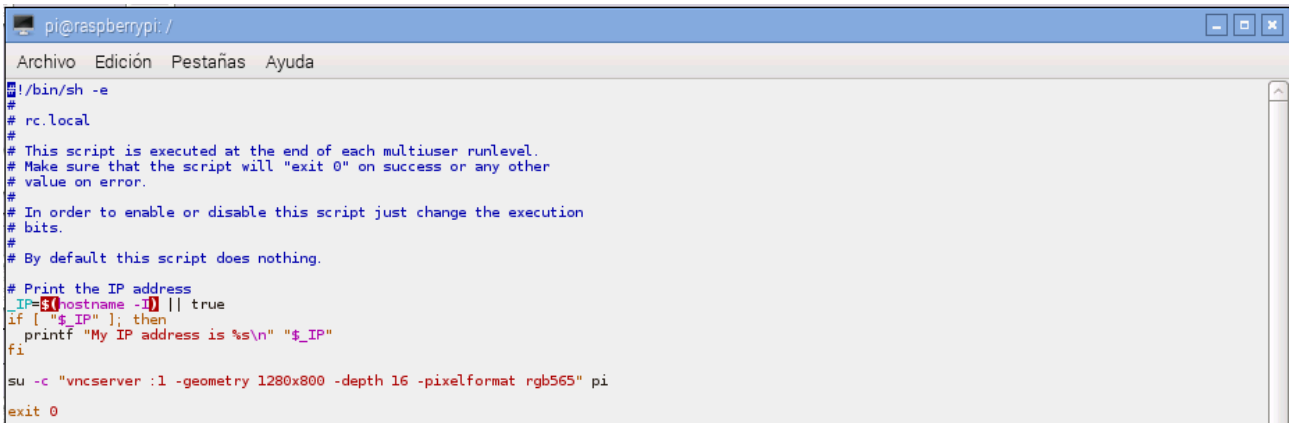


Ilustración 2: Archivo etc/rc.local en Raspberry Pi

A continuación para que los cambios queden guardados se reinicia la

Raspberry con el siguiente comando:

sudo reboot

3º Configuración de red en la Raspberry Pi

Para poder establecer una conexión con la Raspberry debemos realizar determinados cambios de configuración en la tarjeta.

Existen varias maneras de acceder y todas ellas pasan por configurar el fichero "*/etc/network/interfaces*".

Podríamos conectarnos a la Raspberry dejando el servidor dhcp y conectando la tarjeta por cable de red a un router. Tendríamos que obtener la dirección ip por comando desde la terminal de la Raspberry ("*ifconfig* ") pero esto sería engorroso ya que el router podría asignar direcciones ips distintas cada vez.

Auto eth0
iface eth0 inet dhcp

La solución pasa por dos alternativas:

1) Asignar a la Raspberry Pi una dirección IP fija o estática, tras esto, podríamos conectar la Pi a un router o bien directamente por cable de red al ordenador del cliente. Para ello, introducimos las siguientes líneas en el fichero "/etc/network/interfaces":

iface eth0 inet static
address 192.168.1.109
gateway 192.168.1.1
netmask 255.255.255.0

2) Configurar una red wireless. Para poder implementarla necesitamos un adaptador Wifi USB con posibilidad de configuración en modo Ad-hoc que consiste en un tipo de red inalámbrica descentralizada donde todos los nodos tienen el mismo estado dentro de la red y son libres de asociarse con cualquier otro dispositivo de red Ad-hoc en el rango de enlace. El dispositivo Ad-hoc que manejará la Raspberry será el Adaptador Wifi USB TP-Link TL-WN722N a 150 Mbps y antena desmontable de 4 dBi que proporciona mayor potencia de señal al adaptador USB. Para que funcione debemos descargar el fichero htc_9271.fw de su firmware y copiarlo en la siguiente ruta /lib/firmware con el siguiente comando:

sudo cp htc_9271.fw /lib/firmware

Previamente descargaremos el fichero del siguiente enlace de red, con el comando:

sudo wget http://linuxwireless.org/download/htc_fw/1.3/htc_9271.fw

Y es muy importante tener instalados los siguientes paquetes antes de descargar el fichero y copiarlo en el directorio anteriormente indicado:

sudo apt-get install wireless-tools usbutils firmware-atheros

wireless-tools es un paquete que contiene comandos para gestionar redes inalámbricas como poder asociarse a una red con iwconfig, detectar redes con iwlist, etc. Entre sus comandos destacamos iwconfig, iwlist, iwpriv, iwspy, iwgetid e iwevent.

usbutils contiene utilidades para poder inspeccionar los dispositivos conectados al bus USB. Muestra una representación gráfica de los dispositivos que están actualmente conectados, mostrando la topología del bus USB. También muestra información de cada dispositivo que esté conectado al bus.

firmware-atheros contiene los módulos necesarios para habilitar nuestras tarjetas de red inalámbrica y dispositivos bluetooth que dan soporte a los drivers de los dispositivos ar9170, ath3k y ath9k_htc, éste último es el que casa con el dispositivo utilizado en el desarrollo de este proyecto.

A continuación se configura el modo Ad-hoc para el dispositivo en el fichero “/etc/network/interfaces”:

```
auto wlan0
iface wlan0 inet static
address 192.168.1.108
gateway 192.168.1.1
netmask 255.255.255.0
wireless-channel 4
wireless-essid PI_AD_HOC
wireles-mode ad-hoc
```

donde:

- static: Significa que la dirección IP es fija (como indicamos anteriormente).

- address: Establece la IP de la Raspberry.

- gateway: Establece la puerta de enlace.

- netmask: Establece la máscara de subred

- wireless-channel: Establece un canal para la red inalámbrica.

- wireless-essid: Establece el nombre de la red inalámbrica.

- wireless-mode: Establece un modo para la red inalámbrica que en este caso será el modo Ad-hoc. Una tarjeta wireless puede ponerse en modo managed para que trabaje como cliente y necesita para salir a la red un punto de acceso, en modo monitor para estar a la escucha, el modo master permitiendo que otros usuarios en modo managed se conecten a través de tu red o modo ad-hoc definido anteriormente y siempre y cuando, nuestro dispositivo wifi lo permita.

Finalmente el fichero "/etc/network/interfaces" queda como sigue:

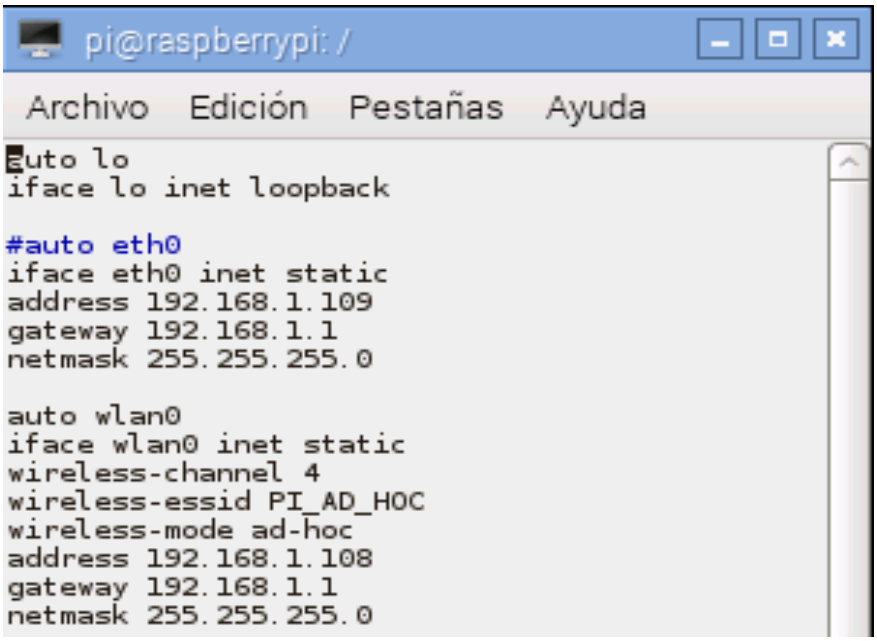


Ilustración 3: Archivo etc/network/interfaces en Raspberry Pi

4º Cliente VNC

Aplicaciones de Cliente VNC hay algunas, tightvnc, ultraVNC pero para este proyecto se ha utilizado " VNC Viewer " de realVNC descargándola desde su página web oficial: http://www.realvnc.com/

Una vez descargada la aplicación para la plataforma que queramos, la ejecutamos:

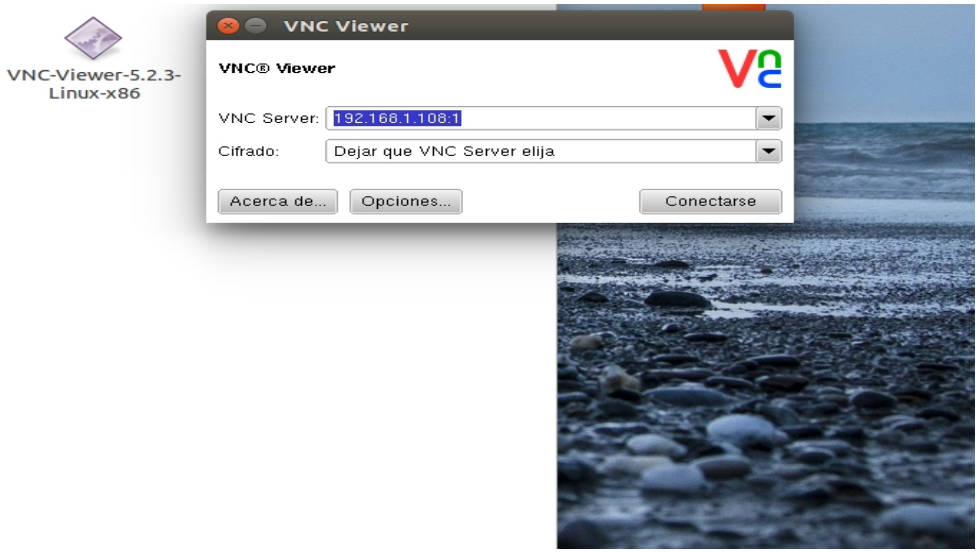


Ilustración 4: Pantalla conexión Cliente VNC con Raspberry Pi

En el campo VNC Server pondremos la dirección IP que hemos configurado para el servidor en el apartado anterior: 192.168.1.108 y seleccionaremos "Dejar que VNC Server elija" para que se la aplicación la que elija el encriptado que quiera utilizar.

Ilustración 5: Ip Raspberry Pi para conectar desde Cliente VNC

Después de pulsar en el botón de "Conectarse" obtendremos el siguiente mensaje en pantalla:

Ilustración 6: Mensaje de conexión no encriptada en Cliente VNC

VNCViewer recuerda que todos los datos que vamos a emplear durante la la conexión que estamos realizando, no serán encriptados. Marcaremos sobre el mensaje de no volver a mostrar este mensaje de nuevo y pulsamos en continuar.

Muy Importante: Es necesario que el ordenador con el que pretendemos conectar a la Raspberry esté dentro de la misma subred donde se encuentra el servidor VNC. Es decir, la dirección IP de nuestro PC con el que queremos conectarnos a la tarjeta debe estar dentro de la numeración 192.168.1.XXX

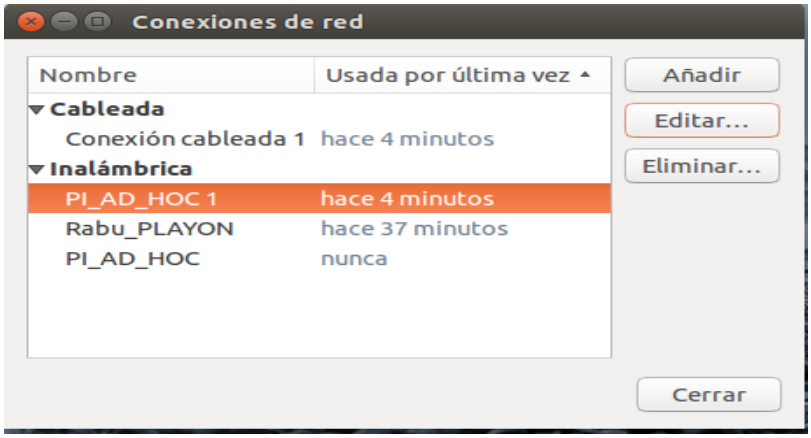


Ilustración 7: Redes disponibles en conexiones de Red en Ubuntu

En Ubuntu la conexión puede configurarse como se muestra a continuación:

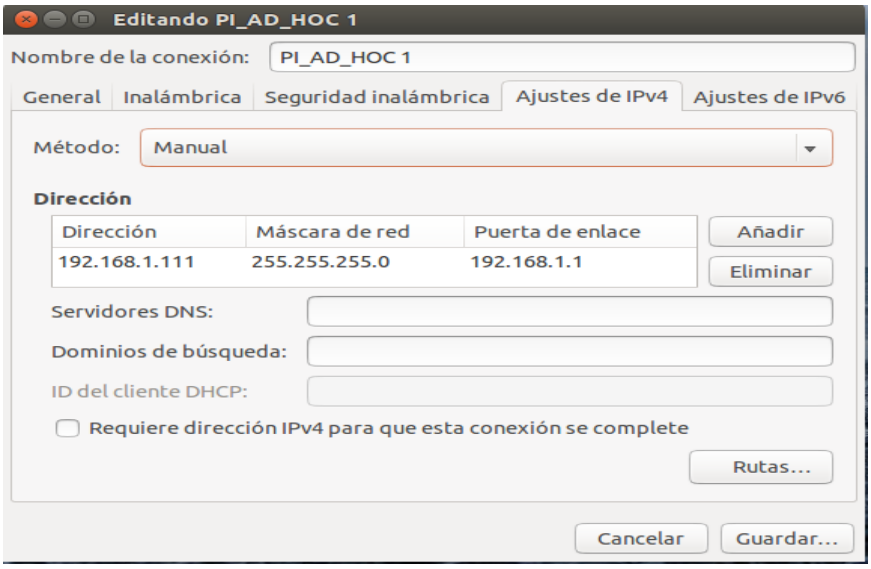


Ilustración 8: Configuración de Ip en Conexión de red PI_AD_HOC

Si todo ha ido bien y las configuraciones se han realizado correctamente, obtendremos una imagen parecida en la pantalla para nuestra Raspberry Pi:

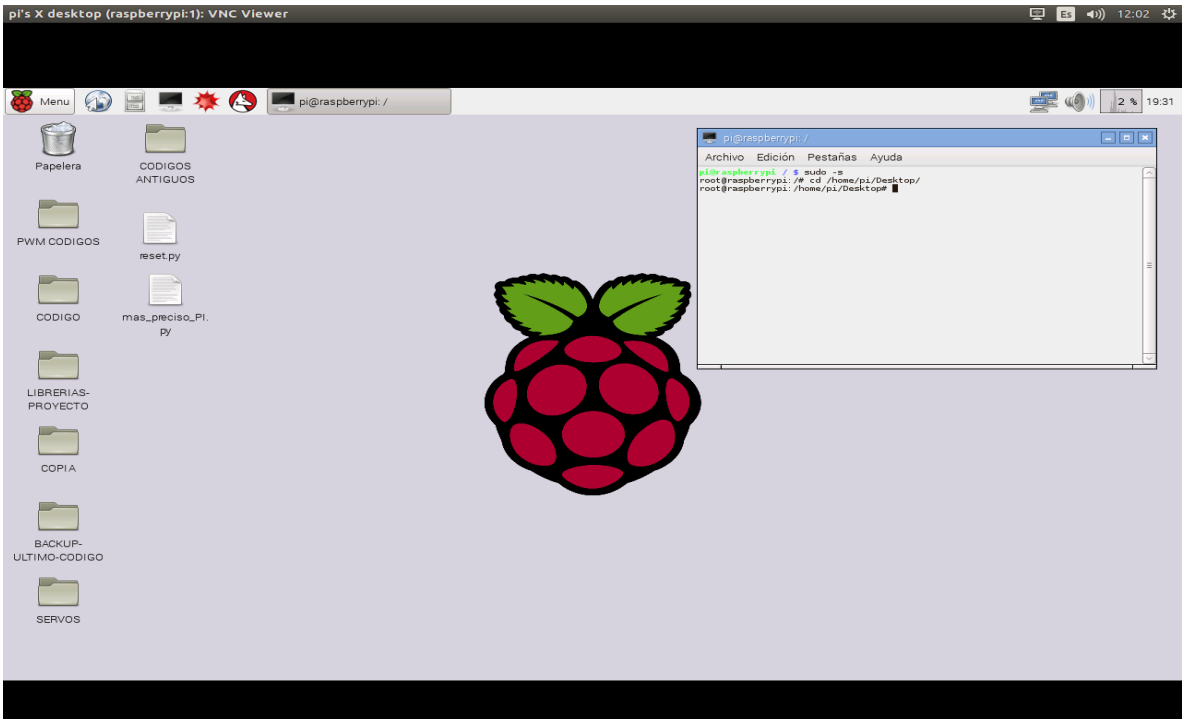


Ilustración 9: Pantalla inicial Raspberry Pi

3.3. Comunicación

La comunicación se encuentra apoyada en el uso de un dispositivo de red Wifi, un adaptador USB inalámbrico y los protocolos TCP/IP (Protocolo de Control de Transmisión y Protocolo de Internet).

El protocolo TCP/IP describe un conjunto de guías generales de diseño e implementación de protocolos de red específicos para permitir que un equipo pueda comunicarse en una red. Es capaz de proveer una conectividad de extremo a extremo especificando cómo los datos deberían ser formateados, direccionados, transmitidos, enrutados y recibidos por el destinatario.

El protocolo TCP/IP presenta las siguientes ventajas:

- Está diseñado para enrutar.

- Tiene un grado de fiabilidad muy elevado.

- Es adecuado para redes grandes y medianas, así como redes empresariales. Esto significa que es muy adecuado para permitir la comunicación entre un elevado número de dispositivos.

- Es compatible con las herramientas estándar que se utilizan para analizar el funcionamiento de una red.

- Puede funcionar en máquinas de cualquier tamaño.

- Soporta múltiples tecnologías.

- Es multiplataforma.

- Está estandarizado puesto que se utiliza a nivel mundial para conectarse a Internet y los a los servidores web.

Sin embargo este protocolo también presenta una serie de desventajas:

- Es difícil de configurar y de mantener.

- El modelo no distingue bien entre servicios, interfaces y protocolos, lo cual afecta al diseño de nuevas tecnologías en base a TCP/IP.

- Es algo más lento en redes con un volumen de tráfico medio bajo, puede ser más rápido en redes con un volumen de tráfico grande donde haya que enrutar un gran número de tramas.

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

SECCIÓN DE INGENIERÍA INFORMÁTICA

Capítulo 4

Módulo de Control

Titulación: Ingeniería en Informática

Alumno: Francisco Raúl Machín Castilla

Tutor: Santiago Torres Álvarez

Julio 2016

Contenido

4 MODULO DE CONTROL

4.1. Introducción

En este módulo se pretende desarrollar el sistema de control necesario para el desarrollo del robot.

Podríamos dividir este módulo de control en dos bloques principales:

- Entradas/Salidas de propósito General (GPIO en Raspberry). En este submódulo se utilizará las librerías necesarias para el manejo y monitorización de las Entradas/Salidas de propósito general para el tratamiento de señales tanto digitales como la PWM.

- Lógica de control. En este submódulo se realizará todo lo referente al control de motores, lectura de los encoders e implementación de código necesario para los movimientos del prototipo.

A continuación se irá definiendo en profundidad en qué consiste cada módulo.

4.2. Entradas/Salidas de propósito general (GPIO).

4.2.1. Descripción

GPIO (General Purpose Input/Output) es un sistema de Entrada/Salida de propósito general, es decir, una serie de conexiones que se pueden usar como entradas o salidas para usos múltiples. Estos pines están incluidos en todos los modelos de Rapsberry Pi y representan la interfaz entre la tarjeta y el mundo exterior. Con ellos se puede hacer multitud de proyectos, desde hacer parpadear un LED hasta muchos otros más sofisticados. Para poder llevar a cabo la tarea de programarlos es muy importante conocer sus características y como se programan. Las características variarán según el modelo y revisión de placa que haya que programar.

A continuación, los pines de la Raspberry Pi según su modelo y revisiones pueden apreciarse en estos esquemas:

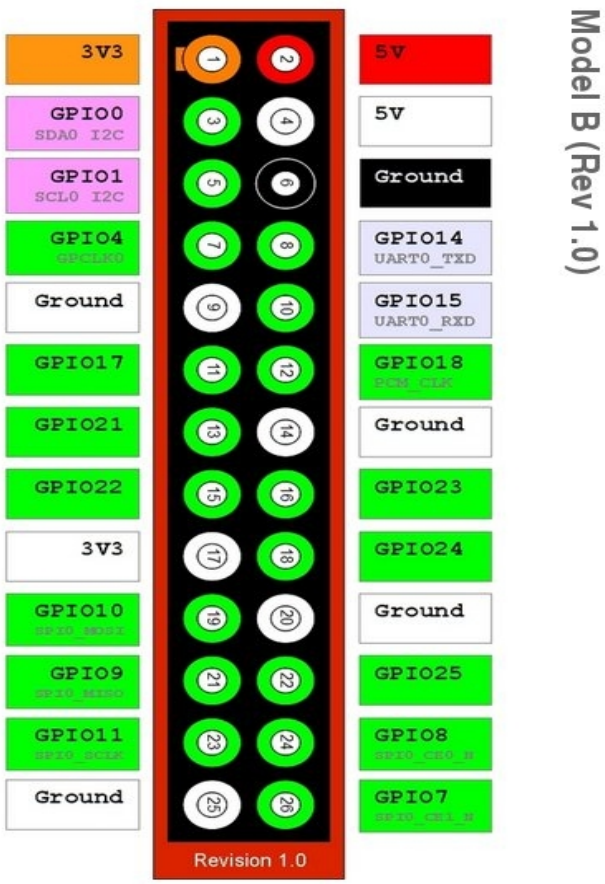


Ilustración 10: Esquema pines GPIO de Raspberry Pi MB Rv1

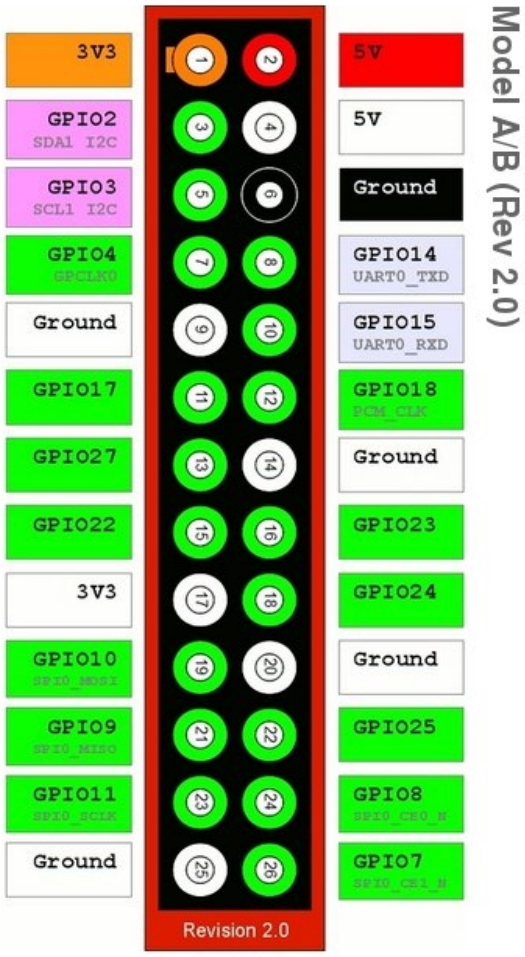


Ilustración 11: Esquema pines GPIO de Raspberry Pi MOD A/B(Rev2.0)

Y para los nuevos modelos de placas:

Ilustración 12: Esquema pines modelos nuevos A+, B+ y Raspberry Pi 2

Los pines de la GPIO de Raspberry Pi son todos del tipo "unbuffered", ésto quiere decir que hay que tener un especial cuidado a la hora de trabajar con ellos porque no disponen de buffers de protección por lo que habrá que tener cuidado con voltajes e intensidades ya que podrían dañar la placa.

Todos los pines no tienen la misma función por lo que están divididos en los siguientes tipos:

- *Pines de Alimentación*.

En los esquemas anteriores se pueden apreciar pines de 5V, pines de 3v3 (limitados a 50mA) y tierra (GND o Ground), que soportan alimentación a estos voltajes para los circuitos que vayamos a implementar. Pueden servir como una fuente de alimentación aunque también se puedan utilizar otras fuentes como pilas, fuentes de alimentación externa, baterías, etc. Son delicadas puesto que al no estar protegidas podrían dañar la Raspberry.

- *DNC (Do Not Connect).*

Son pines que, por el momento no tienen función pero que, en futuras implementaciones muy posiblemente sean utilizados para otros fines. Este es el principal motivo por el que, estos pines, sólo se encontrarán en los modelos más primitivos de Raspberry. En las placas actuales estos pines ya han sido marcados como GND.

- GPIO normales.

Son pines que son configurables y que pueden utilizarse para cualquier proyecto que lo requiera. Estos pines tienen la siguiente descripción: GPIOX ó GPIOXX donde tanto X como XX son números que ayudarán a distinguirlos de forma clara. Existen dos maneras de hacer referencia a los pines de la placa Raspberry, son los sistemas de numeración de los pines: Modo BCM (Broadcom) o Modo BOARD. El modo BCM está refiriéndose a los pines por el número de "canal Broadcom SOC", es decir, usa el número de pin GPIO correspondiente. Por ejemplo, si queremos usar el pin GPIO4, el número 4 será el que accederemos si nos encontramos en el modo BCM. Por el contrario, si la numeración está en modo BOARD, la numeración se basará en el orden de los pines de arriba abajo en la placa (el número de pin físico).

Existen dos tipos de funciones que podemos aplicar a los pines de la GPIO: función de lectura o función de escritura. Sobre estos pines podemos aplicar dos tipos de estados, estado de encendido (HIGH) o estado de apagado (LOW). Informáticamente estos estados los solemos conocer como estado verdadero y estado falso.

- GPIO especiales.

Estos pines pueden estar destinados a diferentes funciones específicas. Hay pines destinados a una comunicación entre dispositivos seriales (pines UART), con conexiones TXD y RXD. Si quisiéramos conectar una placa Arduino utilizaríamos tales pines. Los pines I2C para la comunicación de circuitos integrados, uno de sus usos más comunes es la comunicación entre un microcontrolador y sensores periféricos. I2C es un protocolo síncrono. Utiliza dos líneas SDA(datos) y SCL(reloj). La línea SDA sólo puede cambiar de valor cuando SCL esté a 0. Es posible utilizar un protocolo serie llamado SPI. El funcionamiento de este protocolo es el siguiente: un maestro envia la señal de reloj y tras cada pulso de reloj, envía un bit al esclavo y recibe un bit de éste. Los nombres de esas señales son SPI0_SCLK para el reloj, SPI0_MOSI para configurar el maestro como salida (Out) y el esclavo de entrada (In), y el pin MISO para maestro de entrada (In) y el esclavo de salida (Out). Unidos a estos pines se encuentran los pines CE0 y CE1 que servirán como líneas de control para seleccionar el dispositivo esclavo cuando se envían los datos. El dispositivo seleccionado recibe los datos mientras que el otro dispositivo ignora la comunicación.

4.2.2. Python y Raspberry Pi

Los fundadores de Raspberry Pi recomiendan un lenguaje por su sencillez en la sintaxis y porque puede ayudar bien en asuntos de enseñanza en la educación. Es un lenguaje interpretado o de script, interpretado porque es capaz de realizar la traducción a medida que sea necesaria, típicamente, instrucción por instrucción y, normalmente, no guardan el resultado de dicha traducción. Es decir, con Python, seremos capaces de abrir un entorno donde podremos programar y sobretodo depurar y permiten ofrecer al programa interpretado un entorno no dependiente de la máquina donde se ejecuta el intérprete. Python soporta orientación a objetos, programación imperativa (orientada a procedimientos) y programación funcional, usa tipado dinámico y es multiplataforma.

Lo más importante de Python de cara a Raspberry es que es un lenguaje bastante potente y sobretodo, con muchas librerías ya implementadas que nos ayudarán a realizar casi cualquier cosa.

Python ya viene instalado de serie en las distribuciones que ofrece la fundación Raspberry Pi, sin embargo, si no estuviera instalado por cualquier motivo o bien si quisiéramos instalar otra versión del lenguaje, una de las maneras para poder hacerlo sería la siguiente:

sudo apt-get install build-essential libncursesw5-dev libreadline5-dev libssl-dev libgdbm-dev libc6-dev libsqlite3-dev tk-dev

wget http://www.python.org/ftp/python/3.2/Python-3.2.tar.bz2

donde 3.2 es la versión de Python que queramos instalar. Las diferentes versiones de Python disponibles podemos verlas en este enlace:

https://www.python.org/ftp/python/

descomprimir el fichero descargado:

tar -xjf Pyhton-3.2.tar.bz2 cd Python-3.2/

./configure -prefix=/opt/python3make

sudo make install

Actualmente Python está en la versión 3.6.0

Además, existe una posibilidad en forma de programa llamado “pythonbrew” que nos permite utilizar varias versiones de Python en Linux. Es posible que, en algún momento, tengamos que trabajar con varias versiones de Python en nuestro equipo, para ello, la alternativa más fácil de instalar y usar es “pythonbrew”. Primero se procede a instalar pythonbrew desde el terminal y

para ello existen varias alternativas, una de ellas es utilizar un programa llamado easy_install. Instalar easy_install en Debian se puede hacer utilizando estas líneas desde el terminal:

aptitude install python-dev python-setuptools

A continuación ya podremos instalar pythonbrew:

sudo easy_install pythonbrew

pythonbrew_install

Para instalar las diferentes versiones de Python utilizaremos el comando siguiente:

pythonbrew install 2.7
pythonbrew install 3.2

pythonbrew install X.X (donde X.X corresponde con la versión de Python)

Para seleccionar las diferentes versiones de python utilizaremos el comando:

pythonbrew switch 3.2
pythonbrew switch 2.7

pythonbrew switch X.X

Listaremos las versiones de Python que tenemos instalada con el comando:

pythonbrew list

Y desinstalarlas con el siguiente comando:

pythonbrew uninstall 2.7
pythonbrew uninstall 3.2

pythonbrew uninstall X.X

Para saber si tenemos instalado Python en nuestra Raspberry Pi basta simplemente lanzar desde nuestra terminal el comando:

python

El resultado del lanzamiento del comando anterior debería ser el

siguiente:

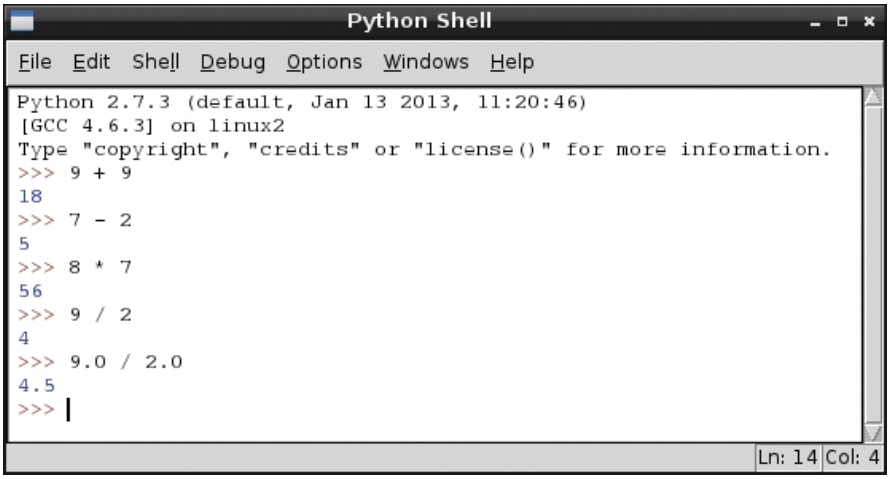


Ilustración 13: shell de Python en Raspberry Pi

Una pantalla donde obtenemos cierta información sobre la versión de Python y una línea de comandos para introducir nuestro código que será interpretado. Como explicamos anteriormente, Python también tiene la posibilidad de lanzar código desde un script utilizando el comando *python "nombre_del_script.py".*

4.2.3. Interactuando con la GPIO en Python

Programar los pines GPIO no es una tarea complicada, para ello, este proyecto utilizará dos librerías llamada Rpi.GPIO (licencia MIT) y RPIO (licencia GPL) que manejan completamente la comunicación con los pines GPIO de la Raspberry. Ambas liberías tienen funciones parecidas sólo que esta última, la RPIO, incorpora un conjunto de herramientas que simplifican y hacen más fácil el manejo de los pines GPIO.

4.2.3.1 Instalación de RPi.GPIO

Existen dos maneras de instalar la librería de comunicación con la GPIO (RPi.GPIO):

1) Instalación desde el respositorio.

Nos bastará con escribir estos comandos en el terminal:

sudo apt-get update (actualiza listado paquetes disponibles)

sudo apt-get install rpi.gpio

ó 2) Instalación Manual.

Sin duda la mejor manera para descargar la librería puesto que elegiremos exactamente que versión queremos descargarnos. Los pasos para instalarla con este método son los siguientes:

- *wget https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.5.11.tar.gz*

Habremos descargado la versión 0.5.11 de la librería que tendrá sus características determinadas.

En la web https://pypi.python.org/pypi/RPi.GPIO encontraremos las diferentes versiones de la librería y las distintas características de cada una.

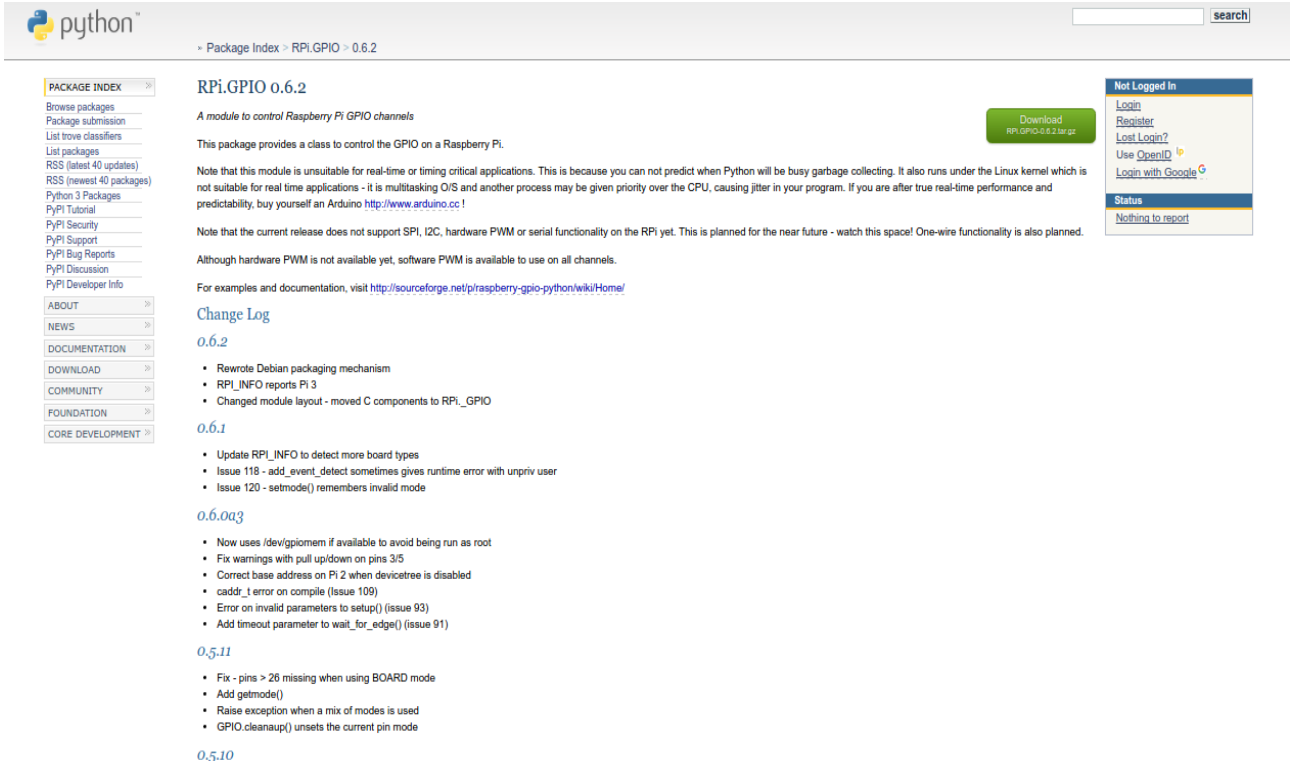


Ilustración 14: Web principal de la librería RPi.GPIO

Si queremos bajar otra versión de la librería basta seleccionar el número de versión en el paquete tar.gz

RPi.GPIO-X.X.X.tar.gz (donde X.X.X corresponde la versión)

Extraer el fichero tar.gz a una carpeta

- *tar -xvf RPi.GPIO-0.5.11.tar.gz*

Entramos a la carpeta que hemos descomprimido:

- *cd RPi.GPIO-0.5.11*

Instalamos la librería haciendo uso del script de python de instalación que viene dentro de la carpeta descomprimida:

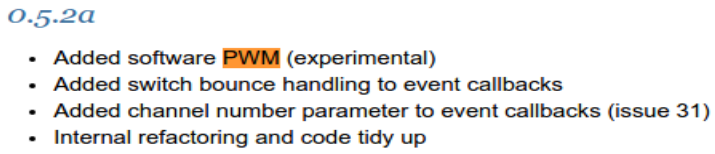
- *sudo python setup.py install*

Por último, procedemos a eliminar la carpeta donde hemos descomprimido los archivos de la librería puesto que ya están instalados:

- *cd ~*

  *sudo rm -rf RPi.GPIO-0.**

A partir de la versión 0.5.2a se incluyó la posibilidad de manejar PWM (Señal Modulada por Ancho de Pulso).



0.5.2a

- Added software PWM (experimental)
- Added switch bounce handling to event callbacks
- Added channel number parameter to event callbacks (issue 31)
- Internal refactoring and code tidy up

Ilustración 15: Ventajas y mejoras de la versión 0.5.2a de Rpio.GPIO

Lo recomendado para esta librería es descargar versiones más recientes puesto que se corrigen errores, se añaden nuevas funcionalidades y mejora la documentación. Actualmente, como indicamos anteriormente la última versión es la 0.6.2 pero nos basta con la que hemos descargado en esta documentación para desarrollar este proyecto.

La librería RPi.GPIO aporta un conjunto de funciones que iremos utilizando durante el desarrollo de este proyecto y que aclaremos más adelante.

4.2.3.2. Instalación de RPIO.

La librería RPIO puede definirse como un "toolbox" o conjunto de herramientas GPIO para la Raspberry Pi. Incorpora PWM via DMA, entradas y salidas por GPIO, interrupciones GPIO, interrupciones socket TCP, herramientas en línea de comandos para visualizar los estados de los pines GPIO en tiempo real (muy útil) y Open Source (LGPLv3+).

La Librería actualmente está en su versión 0.10.0 y su instalación se detalla a continuación:

1) Instalación rápida y sencilla

- *Tras instalar easy_install como explicamos en el apartado 4.2.2 de este documento podremos introducir el siguiente comando:*

sudo easy_install -U RPIO ó

sudo pip install -U RPIO

2) A través de GitHub. **GitHub** es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones **Git**. El repositorio de esta librería se encuentra en GitHub en el siguiente enlace:

https://github.com/metachris/RPIO

La instalación a través de su repositorio en GitHub requiere de tener instalado Git previamente en nuestra Raspberry:

sudo apt-get install git

A continuación, instalamos la librería:

- *git clone https://github.com/metachris/RPIO.git*

- *cd RPIO*

- *sudo python setup.py install*

Podemos encontrar la documentación de esta librería en:

https://pythonhosted.org/RPIO/

Iremos viendo que, a lo largo de esta documentación, se utilizarán determinadas cosas de una librería o de otra. Una de las cosas más importantes por la que se instaló esta segunda librería en la placa es el manejo de una pequeña herramienta desde línea de comandos llamada **rpio-curses**. Esta herramienta nos mostrará en tiempo real el estado de los pines que estamos programando y bastará con ejecutar desde un terminal el comando "rpio-curses" siempre y cuando la librería se haya instalado correctamente.

La herramienta rpio-curses monitoriza los pines GPIO como se muestra continuación:

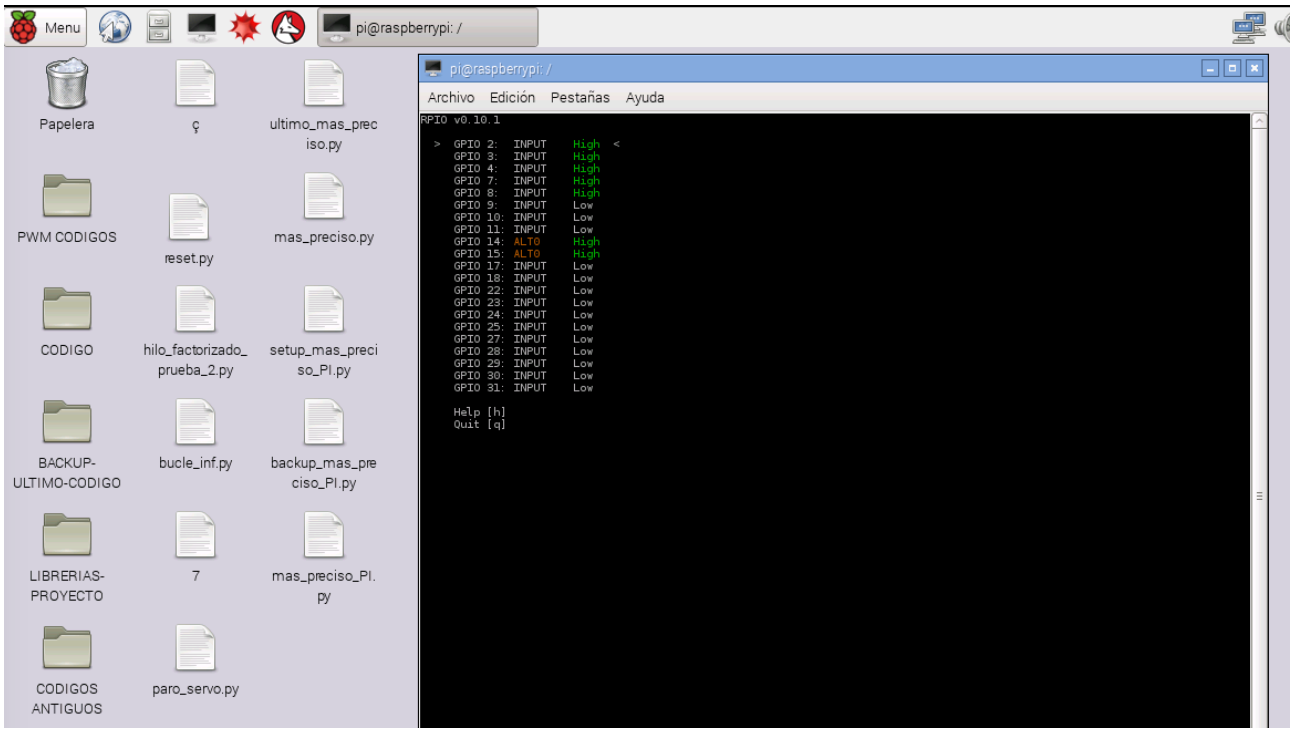


Ilustración 16: Uso de la herramienta rpio-curses en Raspberry Pi

Existe la posibilidad de expandir un menú pulsando sobre la flecha de dirección "->" del teclado:

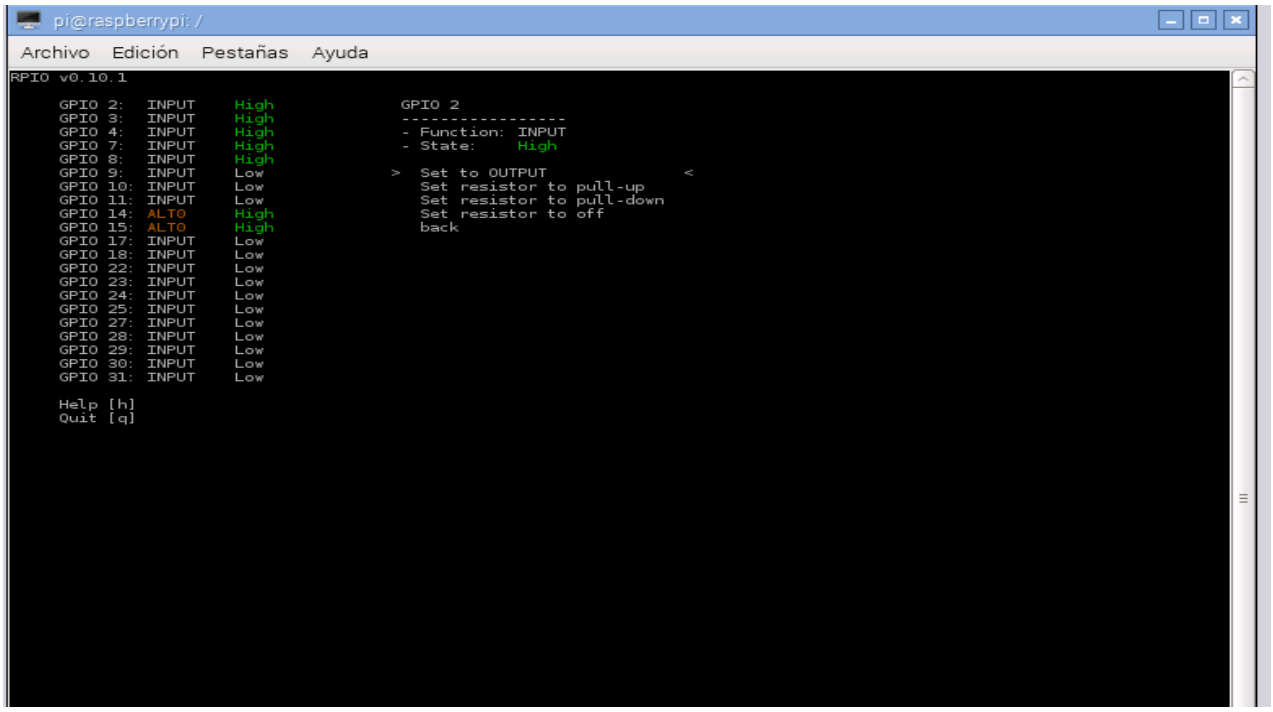


Ilustración 17: Herramienta rpio-curses

En este menú podremos encontrarnos cosas como:

"Set to OUTPUT/INPUT" poner un pin de entrada o de salida

"Set Resistor to Pull/up" *

"Set Resistor to Pull/down" *

"Set Resistor to off" quitar resistencia de los pines

"back" volver al menu anterior.

* A la hora de realizar proyectos electrónicos tenemos componentes que necesitamos que funcionen en dos estados, HIGH o LOW. Pero aunque necesitemos estos dos valores para determinar como actuar es posible que debido a diferentes factores como el ruido eléctrico o variaciones en la fuente de alimentación el valor caiga a un rango indefinido y nos sea imposible determinar si el estado es HIGH o es LOW. Para solucionar esto se utilizan las resistencias Pull-Down y Pull-Up.

La resistencia Pull-Down se conecta a tierra (GND), de esta manera cuando el interruptor este abierto la corriente se dirigida hacia la resistencia dejando un valor 0 en Vout y si el interruptor esta cerrado la corriente se moverá hacia Vout dejando un valor lógico HIGH.

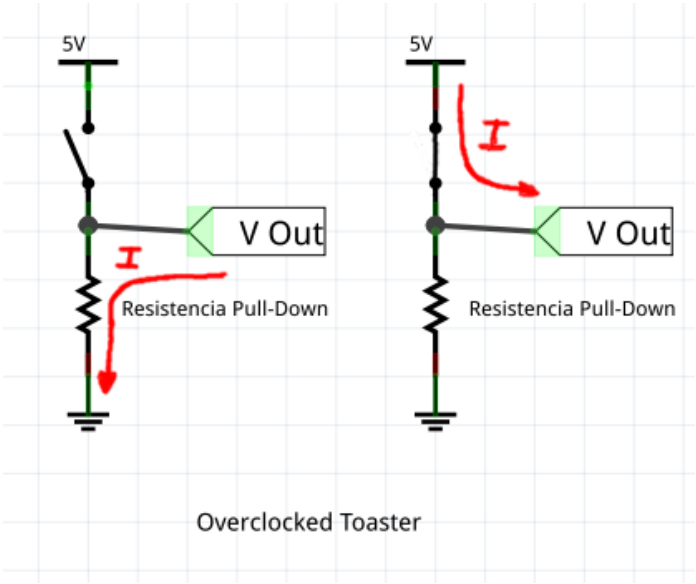


Ilustración 18: Esquema eléctrico resistencia Pull-Down y funcionamiento

La resistencia Pull-Up es lo contrario, cuando el interruptor esta abierto la corriente va desde la fuente de alimentación al Vout dando un valor lógico HIGH y cuando el interruptor esta cerrado la corriente se mueve hacia tierra (GND) dejando un 0 en Vout. Estas resistencias son más limpias y menos propensas a cambiar de estado por señales de alta impedancia.

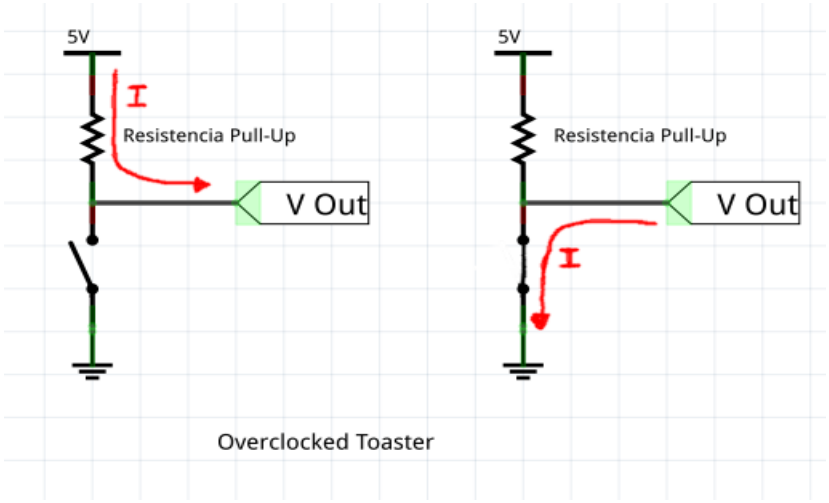


Ilustración 19: Esquema eléctrico resistencia Pull-Up y funcionamiento

Ambas resistencias nos aseguran una lectura correcta en nuestra GPIO. Más adelante explicaremos cómo utilizarlas en los pines a través del código en python.

4.2.3.3. Señal PWM y Python.

La modulación de ancho de pulso o PWM de una señal es una técnica que logra producir el efecto de una señal analógica sobre una carga, a partir de la variación de la anchura de pulso y ciclo de trabajo de una señal digital. El ciclo de trabajo describe la cantidad de tiempo que la señal está en un estado lógico alto, como un porcentaje del tiempo total que este toma para completar un ciclo completo. La frecuencia determina lo rápido que se completa un ciclo, por ejemplo si decimos 100 Hz corresponderá con 100 ciclos en un segundo y por tanto, la frecuencia indica que tan rápido cambia la señal entre los estados lógicos altos y bajos.

El apagado y encendido de un LED puede controlarse con una señal digital (0 y 1) pero no consigue regular el brillo del LED. Para esto se utiliza la señal PWM.

Por tanto, el efecto de aplicar una señal digital PWM sobre un dispositivo de entrada continua o analógica es lograr variar fácilmente el valor

de dicha señal a través de cambios en el ciclo de trabajo de la señal digital. Esto es posible porque el dispositivo en el que se aplica la PWM recibe el valor eficaz de voltaje de la señal digital PWM de alta frecuencia. Dicho valor eficaz es tanto mayor cuanto mayor sea la anchura de pulso de la señal PWM, tal y como se puede observar en la siguiente figura:

Ilustración 20: Variación de Ancho de pulso en PWM

En este proyecto se requiere modificar la velocidad de rotación de los motores del robot móvil Ballbot-ULL. Ambos motores son motores de corriente continua (DC) de 5 Voltios. En el caso de utilizar señal PWM para motores DC precisamos interponer un driver de potencia entre la salida PWM de la placa Raspberry Pi y el propio motor. Más adelante se explicará de qué driver o dispositivo de potencia estamos hablando y su funcionamiento.

En Python para utilizar la señal PWM y tras haber instalado las librerías indicadas en apartados anteriores, se utilizan las siguientes instrucciones:

```
#MOTOR IZQUIERDO = Motor1
#MOTOR DERECHO = Motor2
if (sentido == "ADELANTE"):
        print "ADELANTE"
        Motor1 = 28
        Motor2 = 29

elif (sentido == "ATRAS"):
        print "ATRAS"
        Motor1 = 30
        Motor2 = 31

elif (sentido == "IZQUIERDA"):
        print "GIRO_IZQUIERDA"
        Motor1 = 30
        Motor2 = 29
else:
        print "GIRO_DERECHA"
        Motor1 = 28
        Motor2 = 31

GPIO.setup(Motor1, GPIO.OUT)
p_izq = GPIO.PWM(Motor1, 100)
p_izq.start(0)

GPIO.setup(Motor2, GPIO.OUT)
p_der = GPIO.PWM(Motor2, 100)
p_der.start(0)
```

Ilustración 21: Código de Python para uso de PWM

Tras haber importado la librería RPi.GPIO disponemos el pin correspondiente según que motor queremos mover y creamos una instancia la señal PWM como

var = GPIO.PWM(MotorX, 100) dónde indicamos el pin donde está el motor y 100 Hz de Frequencia.

Iniciamos la variable PWM al 0 por ciento de su “duty cycle” ó ciclo de trabajo. Esto se debe a que inicialmente los motores estarán parados.

```
uk_i = min(uk_i,100)
uk_i = max(uk_i, 0)
p_izq.ChangeDutyCycle(int(uk_i))

uk_d = min(uk_d,100)
uk_d = max(uk_d, 0)
p_der.ChangeDutyCycle(int(uk_d))
```

Ilustración 22: Código de Python para variación de PWM

Con la función “ChangeDutyCycle” que incorpora la librería cambiamos su ciclo de trabajo paulatinamente a los valores indicados de porcentaje en la variable uk_d que explicaremos más adelante su significado.

4.2.3.4. Cables.

Para las conexiones de todos los componentes del robot se utilizarán cables tipo monopin y cables de tipo bus con unas determinadas medidas y conexiones. También, para ciertos componentes como la alimentación de los motores se han utilizado cables normales de cobre con un determinado diámetro.

Los dos tipos de cables más utilizados en este proyecto son los siguientes:

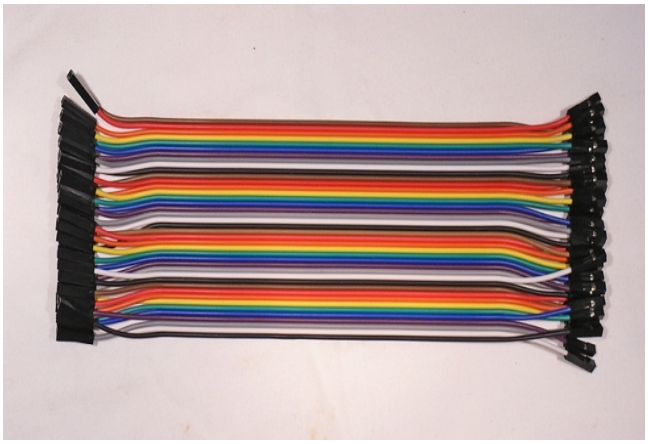


Ilustración 24: Cables monopines H-H

Ilustración 23: Cables monopines M-M

Cables monopines, hembra-hembra, macho-macho, macho-hembra para conexiones individuales de los componentes y sobre la placa protoboard.

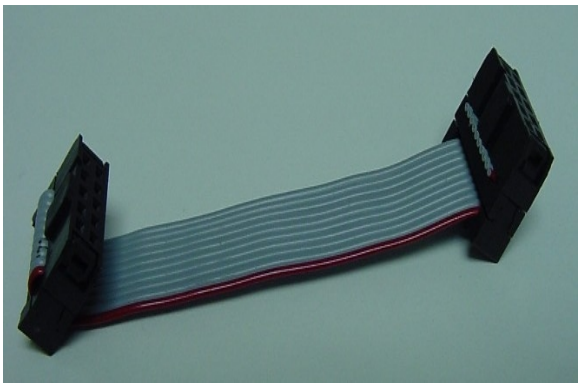


Ilustración 25: Bus Encoders

Ilustración 26: Bus Conexión Raspberry Pi

Cables tipo BUS para conexiones determinadas en los encoders y para la conexión de la Raspberry Pi a la placa protoboard.

4.2.3.5. Circuito de control de motores.

Para poder girar un motor eléctrico en ambos sentidos, avance y retroceso, necesitamos un circuito electrónico que permita realizar dicho cometido. Este circuito se conoce como Puente en H y están disponibles como circuitos integrados pero también pueden construirse a partir de componentes discretos.

Inicialmente el proyecto ya presenta un circuito de control para los motores que hacen desplazar al robot de sitio con suficiente potencia para ello.

El circuito de los motores viene formado por dos controladores de motores doble puente H (L298N) y 8 diodos rectificadores 1N5819 soldados en un circuito hecho a mano más un conjunto de componentes que fueron:

4 Condensadores de 100µF

1 resistencia de 1kΩ

1 resistencia de 330Ω

2 Diodos LEDs (verde y rojo)

1 placa de metal (encargada de disipar el calor en caso que los motores tengan que desplazar cargas pesadas).

Dicho circuito que ya venía implementado tenía este aspecto:

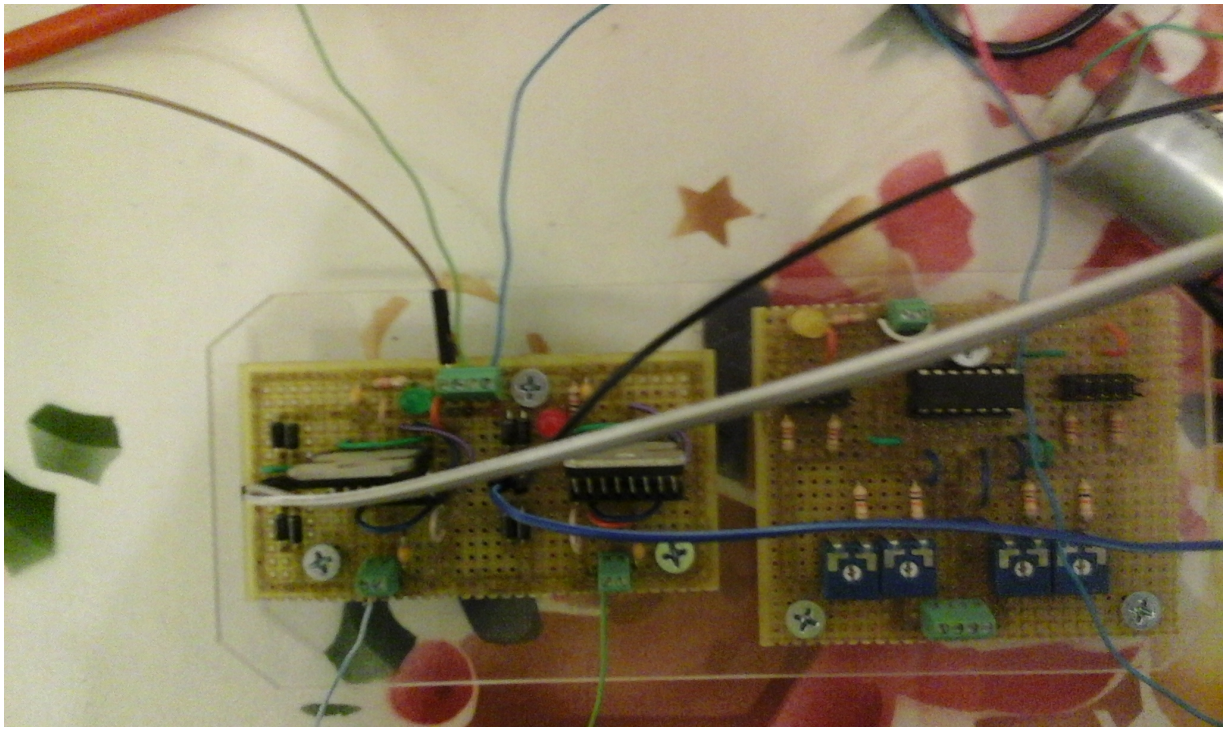


Ilustración 27: Circuito Inicial para Puentes en H a la izquierda imagen

En el proyecto actual, este circuito ya **NO es necesario** y ha sido sustituido por un módulo circuito L298N que ya viene integrado en placa. Dicho circuito es de los más populares entre los aficionados a la electrónica y robótica y que en el mercado no suele supera los 5€ de precio. La decisión de utilizar este dispositivo también persigue un fin de espacio, seguridad y claridad sobre toda la circuitería del robot móvil:

- Ahorra espacio porque su integración es muy aceptable haciendo el dispositivo más pequeño que el circuito hecho a mano que ya presentaba.

- Da más seguridad porque se conectan las entradas y salidas con facilidad y los componentes no quedan al aire pudiendo recibir cualquier golpe.

- Aportan mucha más claridad porque al ser reducidos y utilizar pocos cables, puede visualizarse el resto de circuitería del robot mostrándose más

ordenada. Además aportan menos peso al prototipo, cosa de agradecer.

Utilizaremos dos circuitos ya que cada uno de ellos aporta 2 salidas. Cuatro salidas en total, dos para cada motor (movimiento hacia delante y atrás).

4.2.3.5.1 Circuito integrado L298N.

Este módulo basado en el chip L298N te permite controlar dos motores de corriente continua o un motor paso a paso bipolar de hasta 2 amperios.

El módulo cuenta con todos los componentes necesarios para funcionar sin necesidad de elementos adicionales, entre ellos diodos de protección y un regulador **LM7805** que suministra 5V a la parte lógica del integrado L298N. Cuenta con jumpers de selección para habilitar cada una de las salidas del módulo (A y B). La **salida A** esta conformada por **OUT1** y **OUT2** y la **salida B** por **OUT3** y **OUT4**. Los pines de habilitación son **ENA** y **ENB** respectivamente. En la parte inferior se encuentran los pines de control del módulo, marcados como **IN1**,**IN2**,**IN3** e **IN4**.

El módulo queda inicialmente representado en la siguiente imagen:

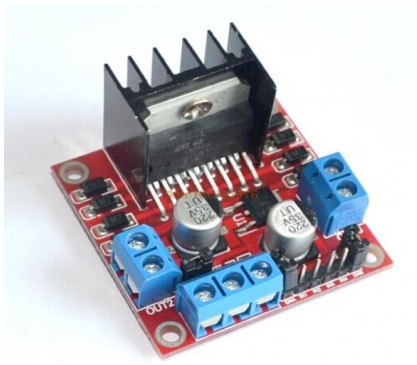


Ilustración 28: Circuito integrado Puentes en H L298N

El módulo inicialmente estará como la anterior imagen pero habrá que realizar algunas modificaciones que permitirán una funcionalidad determinada que explicaremos más adelante.

El módulo L298N posee dos canales de Puente H, pudiéndolos utilizar para controlar dos motores DC o un motor Paso a Paso, controlando el sentido de giro y velocidad. Básicamente está conformado por un driver L298N, sus diodos de

protección y un regulador de voltaje de 5V (78M05).

Posee un conector de 6 pines para ingresar las señales TTL para controlar los motores, un borne de tres pines para la alimentación y dos bornes de 2 pines para la salida de los motores. Esto, en cada parte del circuito queda bien indicado en la siguiente imagen:

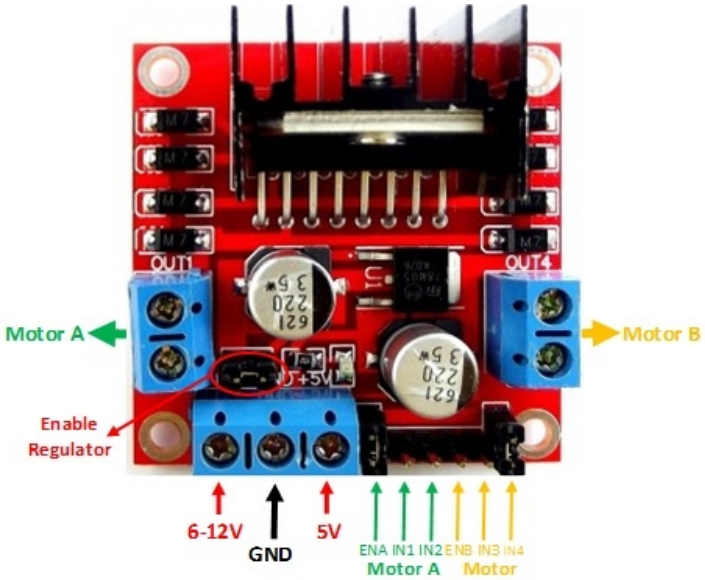


Ilustración 29: Patillaje L298N

Existen dos formas de alimentar el módulo L298N que utiliza este proyecto:

1) Usando una sola fuente, conectada a la entra de 12V y con el Jumper para habilitar el regulador, por supuesto, el voltaje de la fuente es el que soporta el motor. De esta forma, la entrada de 5V no debe estar conectada a ninguna fuente, ya que en este pin están presentes 5V a través del regulador interno; pero puedes utilizar este pin como una salida de 5V, sin excederse de los 500mA. Esta conexión está pensada para voltajes menores de 12V y así no sobrecalentar el regulador.

2) Utilizando dos fuentes, una de 5V conectada a la entrada de 5V (podría ser los 5V de una Raspberry o un Arduino por ejemplo) y otra fuente con el valor del voltaje que trabaja el motor, conectada al pin de 12V. Para ello ha de desconectarse el Jumper, lo que deshabilitará el regulador.

En este proyecto **se ha elegido la primera** forma de conexión para alimentar el módulo L298N.

En el controlador del módulo el funcionamiento es como sigue:

Los pines ENA, IN1, IN2 corresponden a las entradas para controlar el Motor A (OUT1 y OUT2).

De igual manera, los pines ENB, IN3 e IN4 permiten controlar el Motor B (OUT3 y OUT4).

ENA y ENB, sirven para habilitar o deshabilitar sus respectivos motores, generalmente se utilizan para controlar la velocidad, ingresando una señal PWM por estos pines que es lo que hemos explicado en apartados anteriores. Si estos pines, por lo que sea, no se usan, deben estar conectados sus Jumpers puesto que deben estar siempre habilitados.

El esquema eléctrico del módulo L298N es el siguiente:

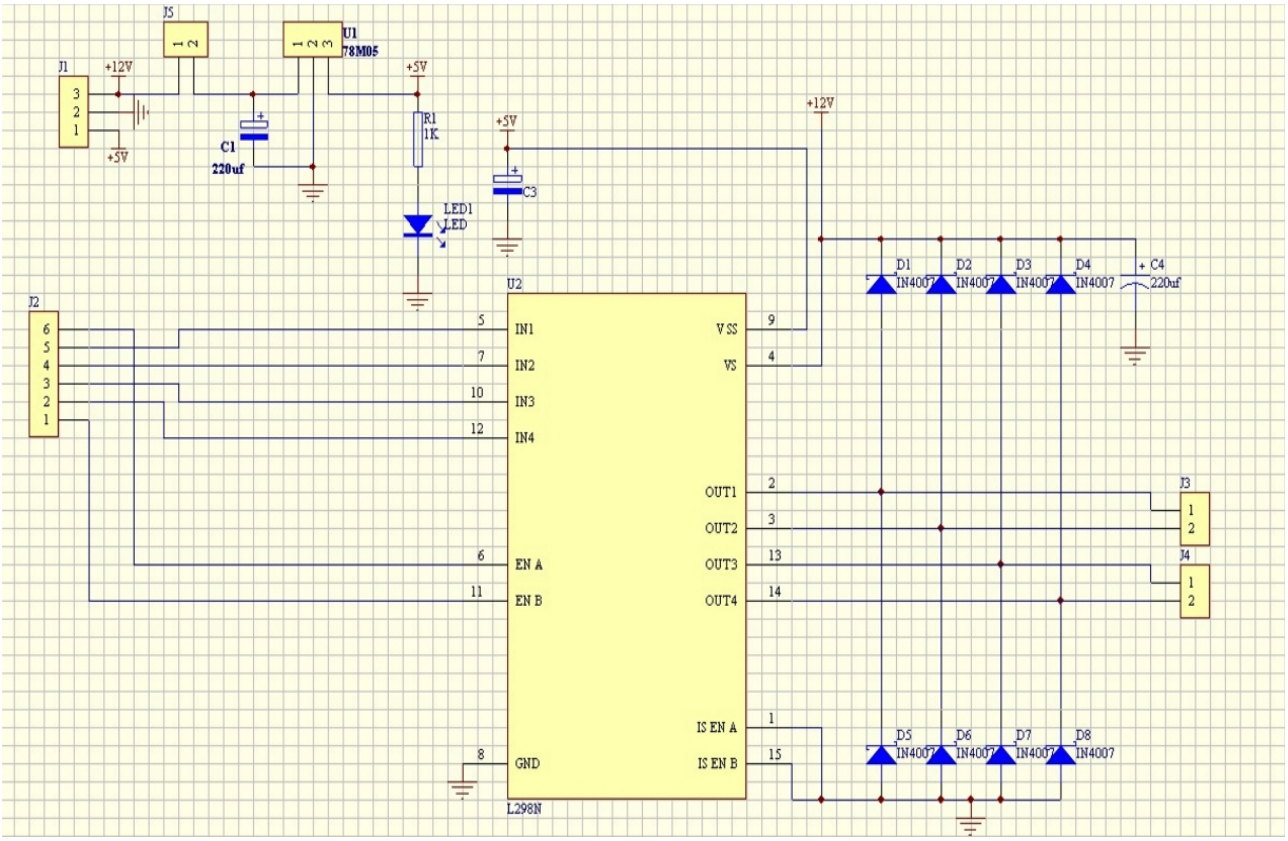


Ilustración 30: Esquema Eléctrico L298N

Una vez conectados los motores a sus salidas y utilizando un puente en H para cada motor ha ocurrido algo con lo que no contábamos al tomar este

camino para desarrollar el proyecto y es que, el módulo L298N **no tiene suficiente potencia** para mover ambos motores a la velocidad deseada. Recordemos que los motores iniciales del proyecto son algo más grandes de tamaño de lo normal y han de mover una carcasa de metacrilato que depositará peso sobre y dentro de ella.

Para solventar este problema debemos conectar en paralelo ciertas entradas y salidas. En el siguiente esquema podemos ver cómo qué salidas y qué entradas debemos conectar para conseguir el doble de potencia:

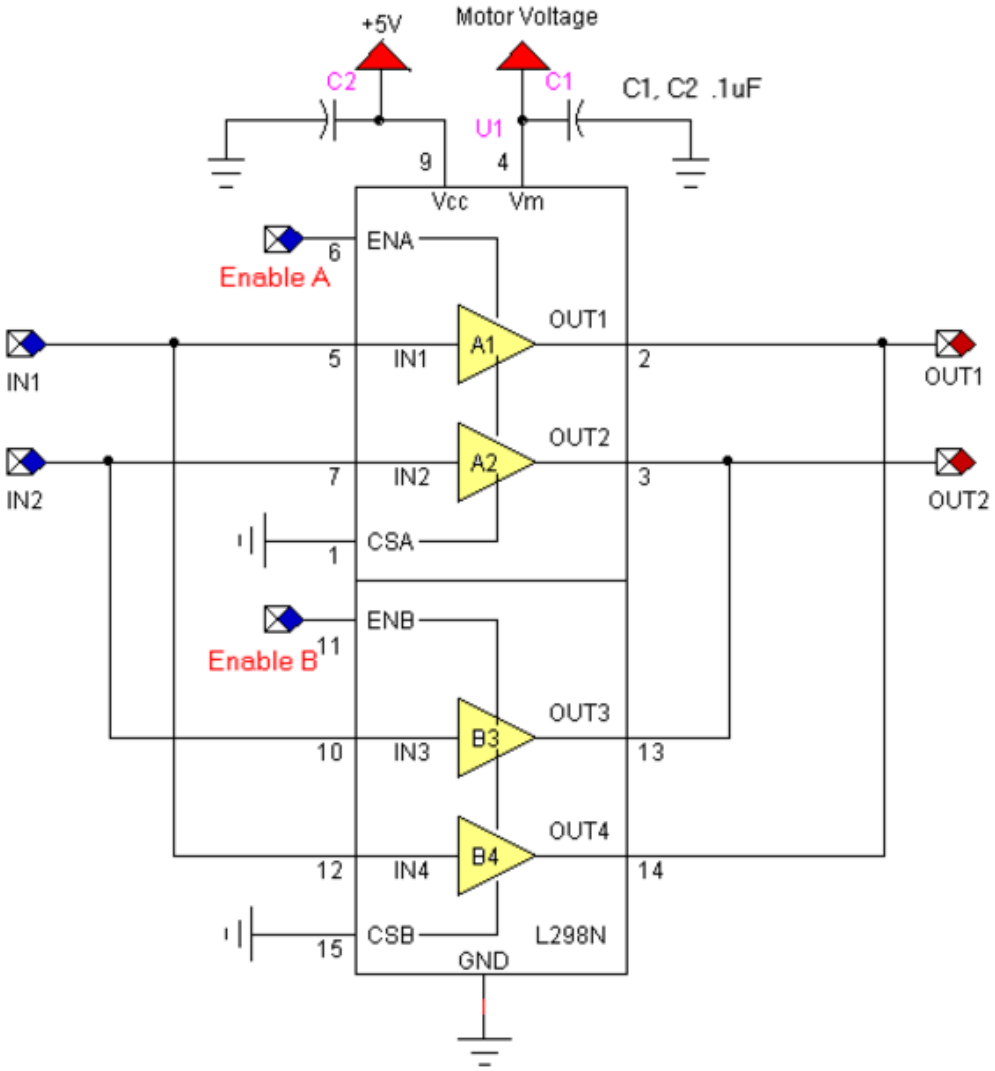


Ilustración 31: Esquema Eléctrico L298N para conexión en Paralelo de entradas-salidas

En el esquema puede observarse la conexión entre las entradas y las salidas. Basta con conectar las entradas IN1 y la IN4, las entradas IN2 e IN3.

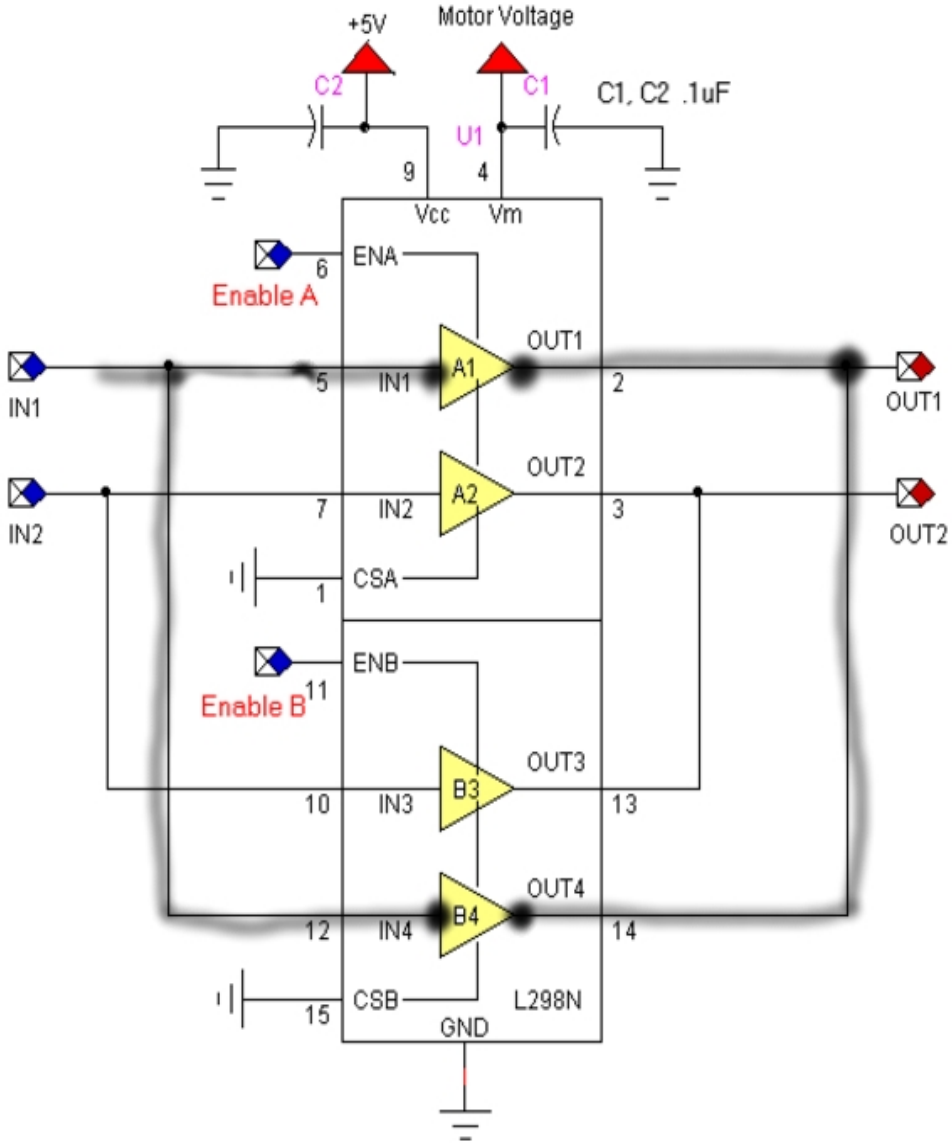


Ilustración 32: Conexión Entradas 1 y 4 – Salidas 1 y 4 L298N

También hay que interconectar entre sí las salidas OUT1 y OUT4 y las salidas OUT2 y OUT3.

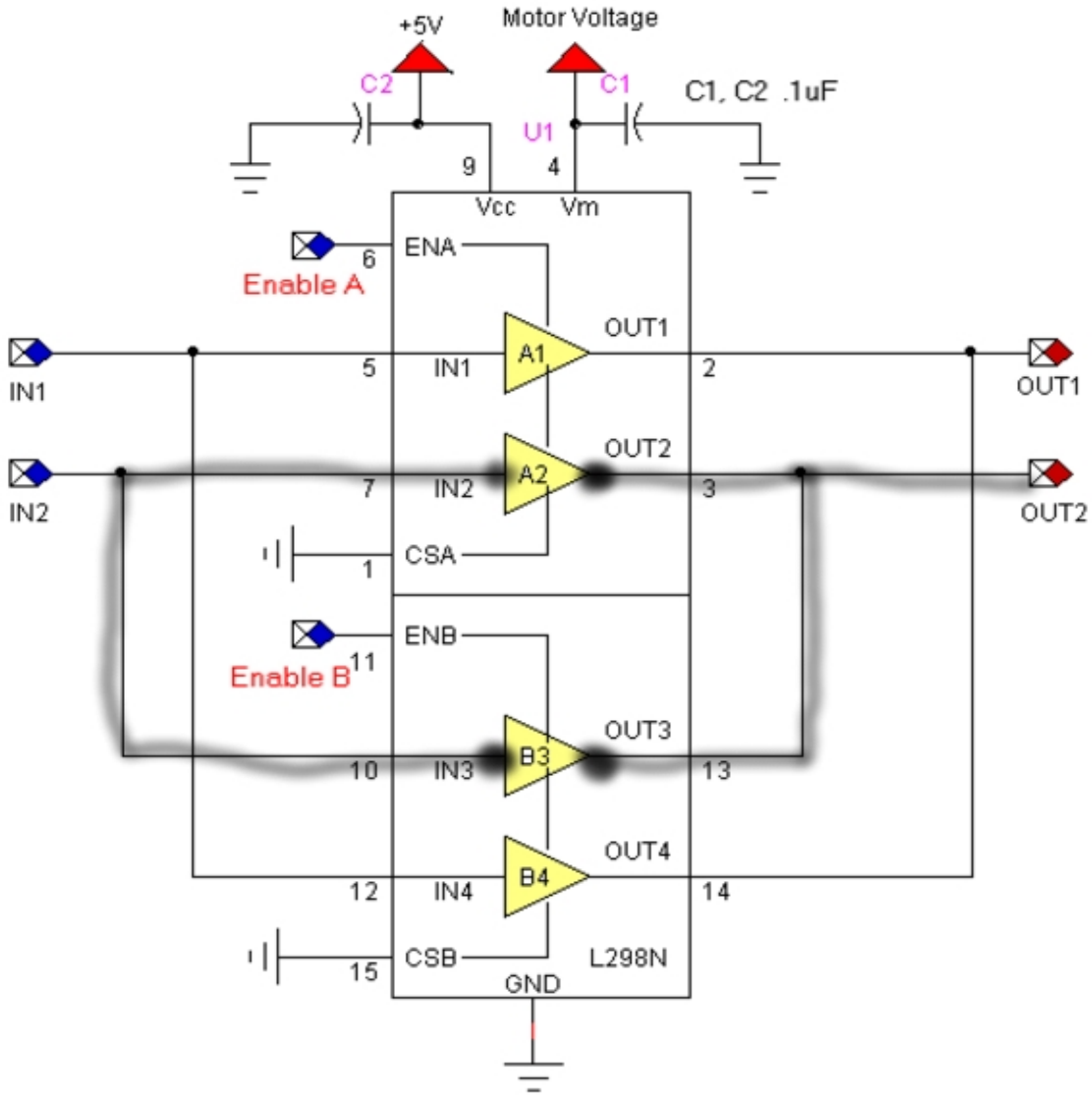


Ilustración 33: Conexión Entradas 2 y 3 – Salidas 2 y 3 L298N

Por lo tanto hay que realizar una modificación al módulo L298N de la mejor manera posible para que nos permita utilizar esta posibilidad que se ha encontrado en la documentación del controlador. Dicha información ha sido obtenida en el siguiente enlace: http://www.bristolwatch.com/L298N/

Puede observarse que dicha documentación no es exactamente para el módulo que utilizamos en el proyecto ya que dicho módulo es diferente en su diseño físico:

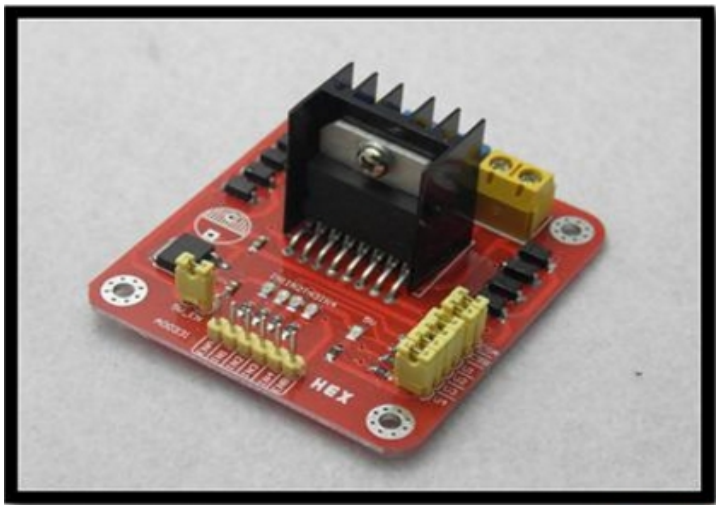


Ilustración 34: Otros modelos circuito L298N

Sin embargo, su diseño lógico es el mismo por lo que procedemos a conectar las entradas y salidas indicadas. El resultado tras realizar la soldadura lo más limpia y clara posible en el módulo, ha quedado de la forma siguiente:

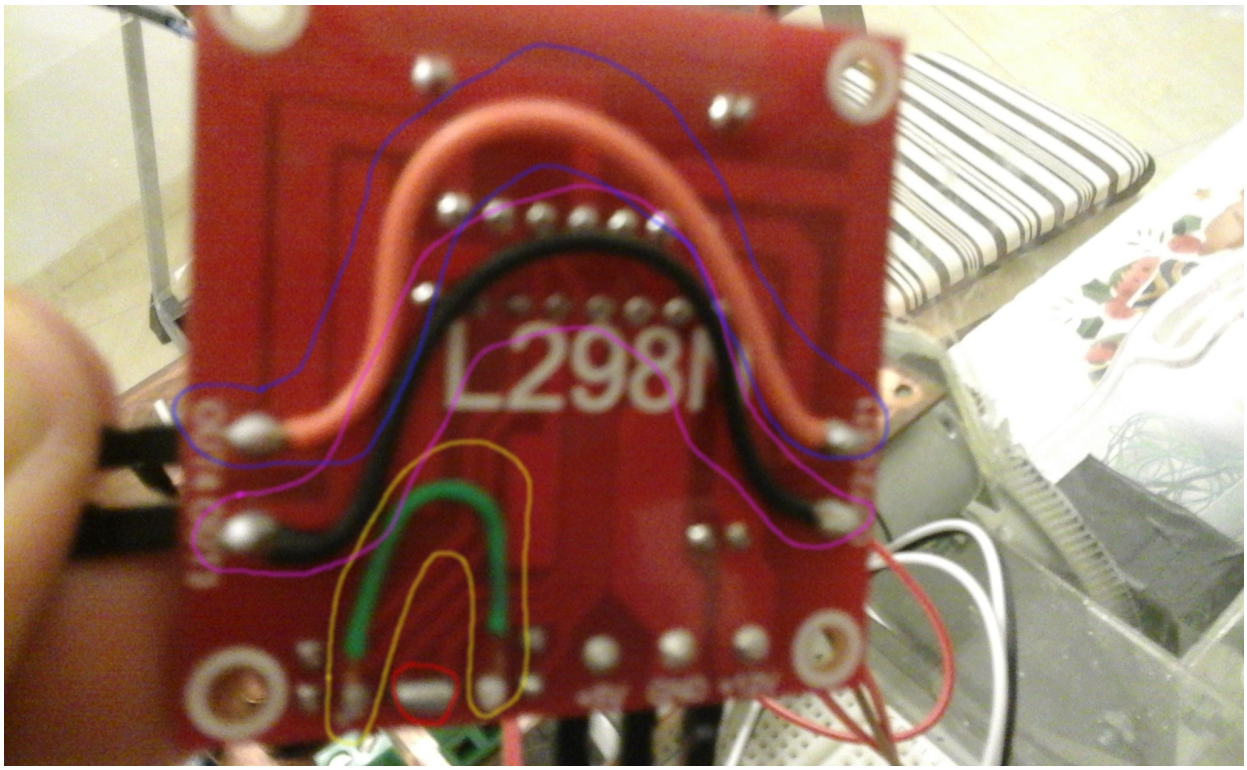


Ilustración 35: Soldadura conexiones Entradas – Salidas L298N

Sobre la imagen, las líneas de colores en las soldaduras representan lo siguiente:

- Círculo Rojo: soldadura Entradas IN2 e IN3

- Cable color verde dentro de línea amarilla: soldadura Entradas IN1 e IN4

- Cable negro dentro de línea violeta: soldadura Salidas OUT2 e OUT3

- Cable naranja dentro de línea color azul: soldadura Salidas OUT1 e OUT4

Deben soldarse de esta forma ambos módulos L298N, ya que utilizaremos como hemos dicho antes, uno para cada motor. Tras esto, ya tenemos el doble de potencia lista para aplicarla sobre los motores.

Finalmente, la conexión de los dispositivos L298N en nuestro robot móvil,

queda como recoge la siguiente fotografía:

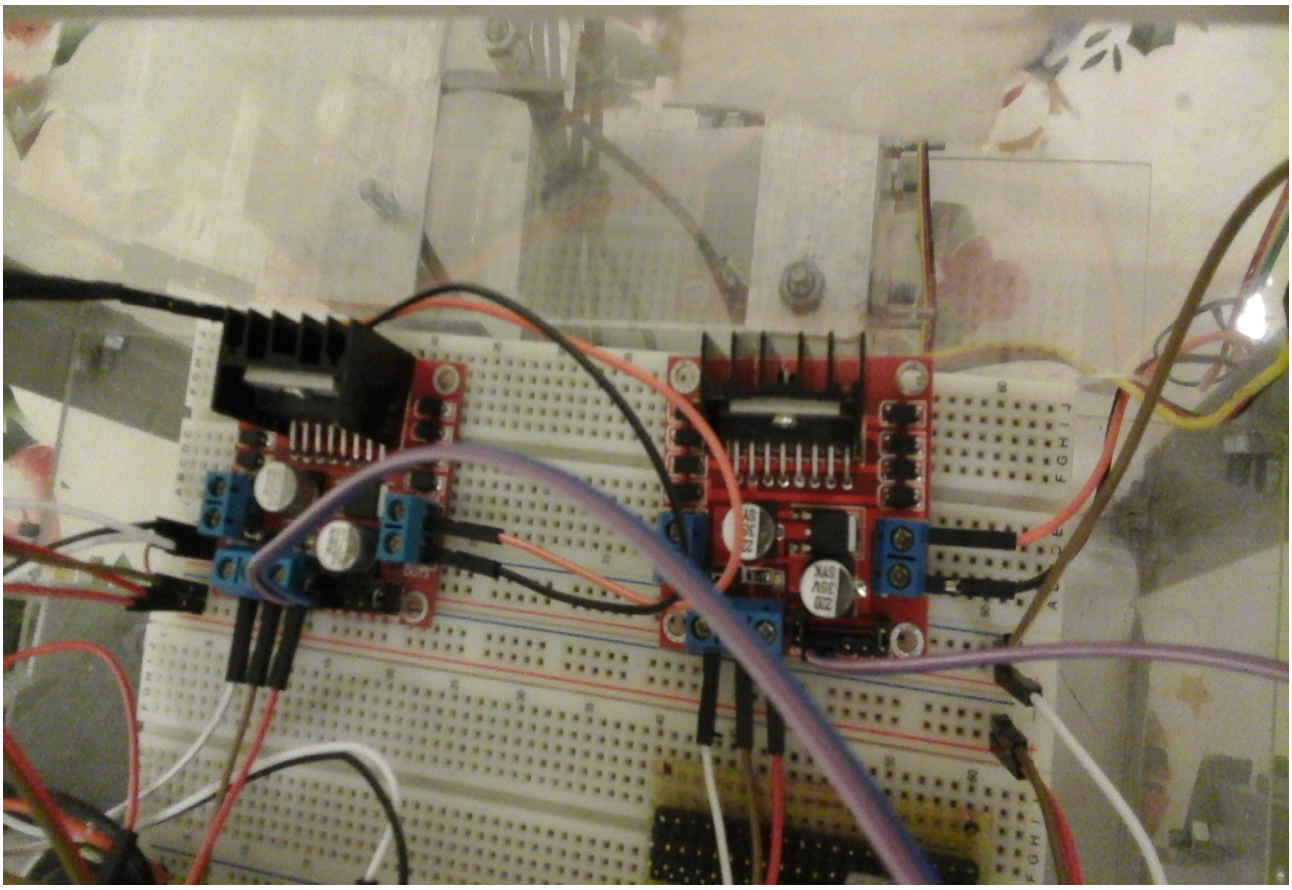


Ilustración 36: L298N conectados en el prototipo

El módulo L298N se alimenta de la primera forma comentada anteriormente. Estamos introduciendo una sóla fuente de alimentación que, en este caso, es la pila que nos surte de 11.1 Voltios y al mismo tiempo, a través de un dispositivo que hará de “fuente intermedia” ó de regulador de tensión y que, más tarde explicaremos con detalle, utilizamos la alimentación de 5 Voltios.

4.2.3.6. Eliminación de Circuito para la comunicación Serial.

Inicialmente el prototipo trae consigo un circuito para la comunicación serial entre la Raspberry Pi y el Arduino UNO. Dicha circuitería viene soldada en placa como puede apreciarse en las siguientes imagenes:

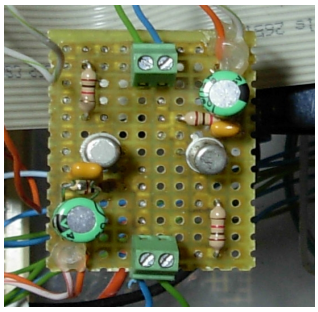


Ilustración 37: Circuito comunicación Serial

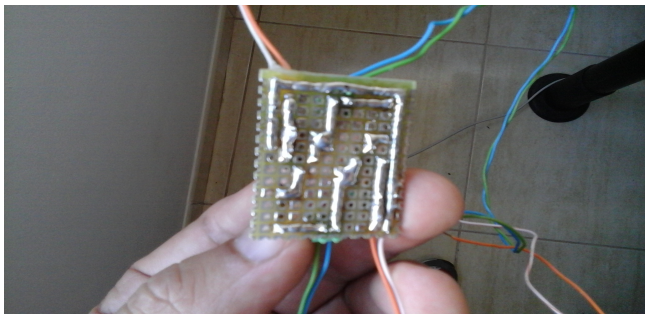


Ilustración 38: Soldaduras circuito Serial

Arduino UNO era el microcontrolador que manejaba los motores y los encoders liberando así de ese trabajo a la Raspberry Pi y por otro lado, ralentizando las acciones de movimiento del robot.

El circuito iba conectado entre las dos placas como indica el siguiente esquema eléctrico:

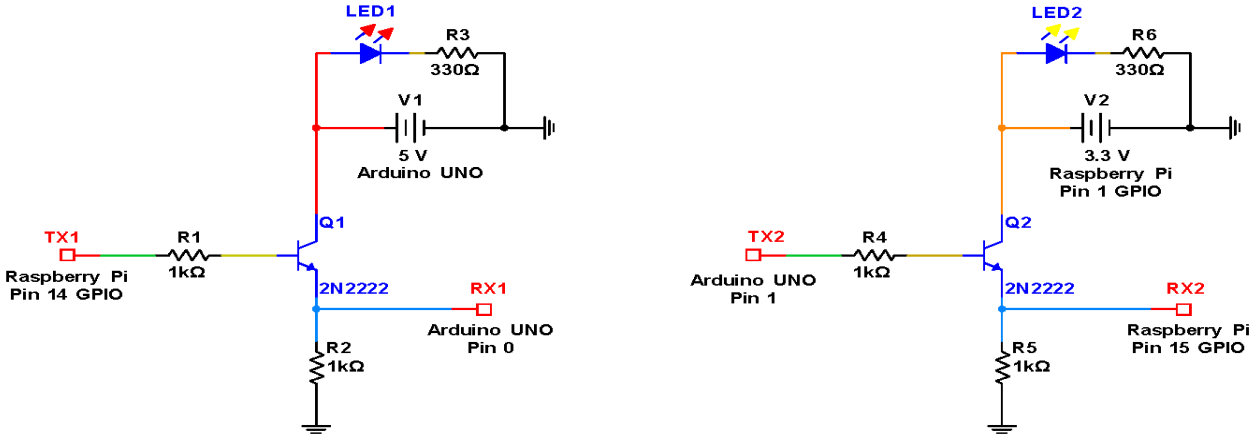


Ilustración 39: Esquema Eléctrico del Circuito Serial de comunicación entre Arduino UNO y Raspberry Pi que fue suprimido del proyecto actual

Es decir, las conexiones iban desde la GPIO de la Raspberry en los pines 14 TXD (Envío de señal) y 15 RXD (Recepción de señal), habilitados en la GPIO para este tipo de cometido (comunicación serial) y los pines 0 y 1 del Arduino UNO respectivamente.

El problema que tiene Raspberry Pi en las placas antiguas es que el puerto GPIO utiliza 3.3 voltios y no tienen ningún tipo de protección contra sobrevoltajes. Por este motivo, si se utiliza directamente los voltajes de la placa Arduino UNO de 5 Voltios, se corre el riesgo de dañar los puertos y quemar la Raspberry Pi. Para evitar que esto ocurra, se pueden utilizar transistores y resistencias para igualar los niveles de voltaje y no quemar los circuitos de ambas placas. Es en esto precisamente, en lo que se basa el circuito de comunicación serial y que recordemos que ahora, **ya no es necesario puesto que la Raspberry se encargará de sobrellevar toda las tareas** de movimiento de motores, lectura encoders, servos, control, etc.

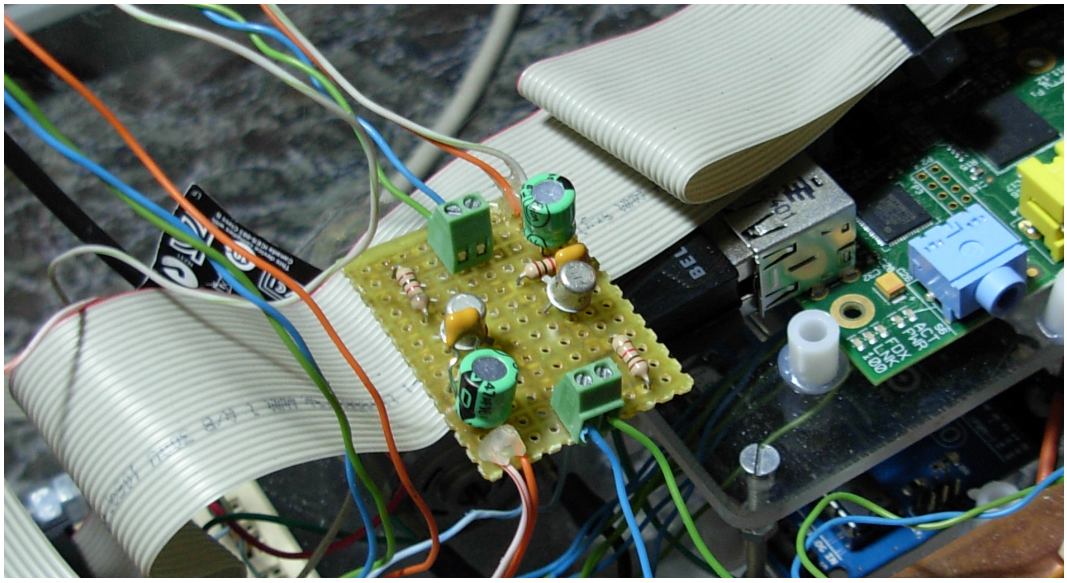


Ilustración 40: Circuito Serial inicial conectado entre Raspberry Pi y Arduino UNO

4.2.3.7. Circuito para la lectura de los encoders.

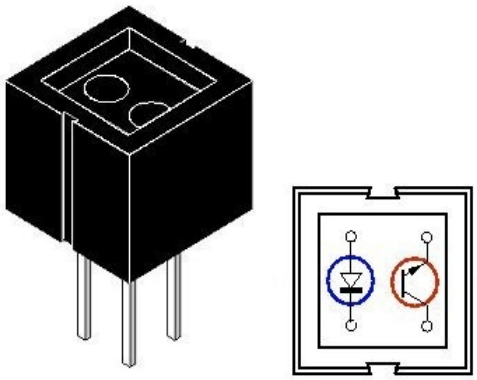
Un ***encoder óptico*** es un sensor que permite detectar el movimiento de rotación de un eje. En definitiva, se trata de un transductor que convierte una magnitud de un mecanismo, tanto posición lineal como angular, a una señal digital. El encoder estará operando en relación al eje del elemento cuya posición deseamos determinar. Su fundamento viene dado por la obtención de la medida en base a la luz que traspasa una serie de discos superpuestos que codificarán la salida digital.

El principio del funcionamiento de un encoder se basa en los llamados fotoacopladores. Éstos son pequeños chips que consisten en un diodo en

forma de fotoemisor y un transistor que realiza las tareas de fotorreceptor. Este elemento se encarga de detectar la presencia/ausencia de luz a través de los discos concéntricos al eje, los cuáles, en este caso en concreto, están "marcados" con unas cintas que los rodean, una en cada rueda del robot móvil, con colores alternados de blanco y negro. Esos cambios de luz se codificarán para obtener una medida determinada.

El encoder utilizado en este proyecto es el sensor CNY70.

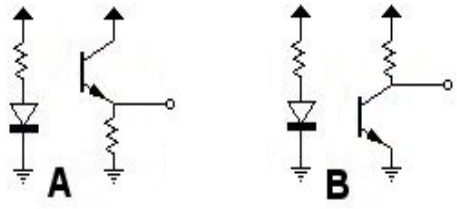
El CNY70 es un sensor de infrarrojos de corto alcance basado en un emisor de luz y un receptor, ambos apuntando en la misma dirección, y cuyo funcionamiento se basa en la capacidad de reflexión del objeto, y la detección del rayo reflectado por el receptor.



Vista externa y circuitos internos del sensor CNY70

Ilustración 41: Vista externa y ciruitería CNY70

El sensor tiene cuatro pines de conexión. Dos de ellos se corresponden con el ánodo y cátodo del emisor, y las otras dos se corresponde con el colector y el emisor del receptor. Los valores de las resistencias son típicamente 10K ohmios para el receptor y 220 ohmios para el emisor.



Diferentes posibilidades de montaje del CNY70

Ilustración 42: Montaje CNY70

El CNY70 devuelve por la pata de salida correspondiente, según el montaje, un voltaje relacionado con la cantidad de rayo reflectado por el objeto. Para el montaje A, se leerá del emisor un '1' cuando se refleje luz y un '0' cuando no se refleje. Para el montaje B los valores se leen del colector, y son los contrarios al montaje A.

Si conectamos la salida a una entrada digital del microcontrolador, entonces obtendremos un '1' o un '0' en función del nivel al que el microcontrolador establece la distinción entre ambos niveles lógicos. Este nivel se puede controlar introduciendo un buffer trigger-schmitt, por ejemplo, el HCF40106BE, entre la salida del CNY70 y la entrada del microcontrolador. Este sistema es el que se emplea para distinguir entre blanco y negro, en la conocida aplicación del robot seguidor de línea.

Otra posibilidad es conectar la salida a una entrada analógica. De este modo, mediante un conversor A/D se pueden obtener distintos valores. Esto permite la detección dinámica de blanco y negro (muy útil cuando el recorrido presenta alteraciones en la iluminación). Pero también, si empleamos el sensor con objetos de distintos color, establecer un mecanismo para la detección de los distintos colores, determinando los valores marginales que separan unos colores de otros. Esto permite emplear el sensor para alguna aplicación donde la detección del color sea necesaria.

Es importante conocer la orientación del encoder para saber cómo conectarlo correctamente, las siguientes fotografías fueron obtenidas con ese fin, saber la orientación del encoder al introducirlo en las ranuras para medir los cambios de línea (blanco y negro) en la rotación de sus ruedas:

Ilustración 43: Vista frontal encoder

Ilustración 44: Vista lateral encoder

Es importante fijarse en las letras escritas en el lateral del encoder, nos ayudará a conocer bien la orientación del mismo a la hora de "conectarlos" a las ruedas.

Una de las desventajas más importantes que presenta este sensor CNY70 es que necesita estar bastante próximo a la cinta donde necesita medir ya que su halo de luz y la recepción de ese reflejo de luz para la lectura no son tan potentes.

Estos dos circuitos integrados por sí solos no bastan y hay que montar una circuitería capaz de realizar la lectura de los encoders, para ello, a parte de usar 1 HCF40106BE y 4 encoders CNY70, necesitamos lo siguiente:

- 4 Potenciómetros de 22kΩ
- 4 Resistencias de 10kΩ
- 4 Resistencias de 220Ω
- 1 Resistencia de 330Ω
- 2 Diodos LED (Verdes)
- 2 Diodos LED (Amarillos)
- 1 Diodo LED (Rojo)

El esquema eléctrico del circuito es el siguiente:

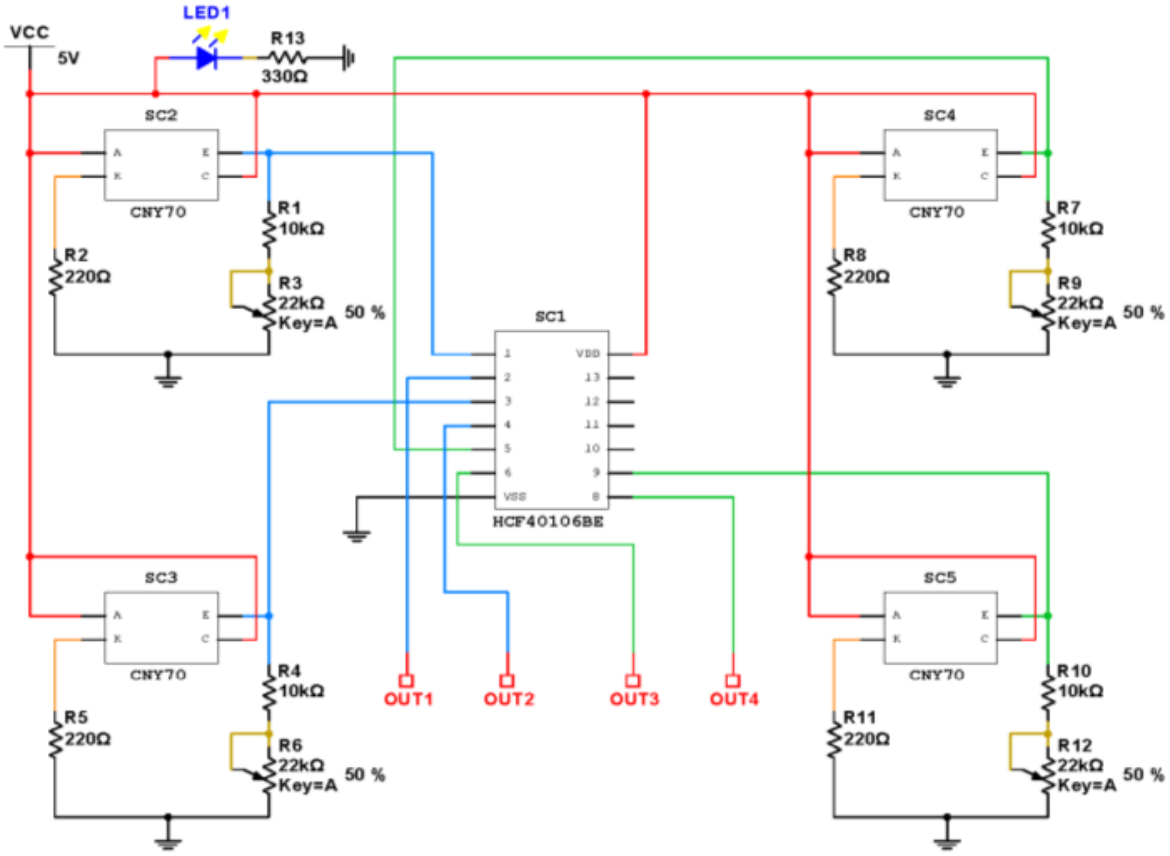


Ilustración 45: Esquema eléctrico del circuito de lectura de encoders

En el circuito anterior se puede observar 4 Salidas (OUT1, OUT2, OUT3 y OUT4), las salidas OUT1 y OUT2 son las de los encoders que obtienen información de la rueda izquierda del robot, las salidas OUT3 y OUT4 son las de los encoders que obtienen información de la rueda derecha del robot. El LED amarillo se utiliza para indicar que el circuito esta alimentado. El circuito es alimentado con 5 Voltios.

Para conocer el sentido de giro a través de los encoder polarizamos adecuadamente el CNY70 para que siempre esté emitiendo luz y orientándolo contra una superficie blanca, produce un reflejo que será recibido por el fototransistor, quedando éste también polarizado y devolviendo un valor cercano a 5 Voltios. Por el contrario, si la superficie es negra, se absorve la luz infrarroja y no produce reflejo hacia el fototransistor con lo que el valor medio en su salida es cercano a los 0 Voltios. Por este motivo se usa el "Schmitt trigger" para conseguir que los estados sean completamente 1 o completamente 0 y no se queden en un estado intermedio.

El prototipo para este proyecto ya viene inicialmente con una superficie circular contra el CNY70 de manera que alterne franjas radiales negras y blancas

(en total 12 franjas para el diseño de este proyecto), al girar en un sentido (sobre un eje) solidariamente al movimiento de rotación del elemento del cual queremos medir su posición angular, notaremos los cambios proporcionales producidos en la salida del fototransistor y que podemos contabilizar por software, obteniendo así la modificación de la posición en cada momento.

Disco de franjas para codificación del sensor CNY70.

Ilustración 46: Discos con marcas para codificación

Si el movimiento de rotación que queremos precisar es en ambos sentidos (como ocurre en este caso), necesitamos añadir otro CNY70 contra el disco de forma que estén desplazados una cantidad tal que generen una secuencia de pulsos de salida desfasados 90 grados según emitan contra superficie blanca o negra.

Además se añade un circuito integrado con varios disparadores Schmitt (trigger Schmitt) en configuración inversora, que consiguen mejorar la señal invirtiendo su valor por medio de un ciclo de histéresis en cada salida de su correspondiente fototransistor. Con ello conseguimos que los niveles se estabilicen y adapten para ser conectados a una entrada digital y ser interpretados con mayor precisión.

Por lo tanto, por cada “encoder casero” tenemos dos CNY70 con cada una de sus salidas (del fototransistor) conectadas a un disparador Schmitt inversor y una superficie circular con franjas radiales negras y blancas intercaladas entre sí, enfrentada a los CNY70. Cada uno de los dos CNY70 lo trataremos como un canal de señal. A este tipo de encoders con dos canales se les conoce como encoder de cuadratura y generan dos secuencias digitales distintas en función de que el giro se realice en un sentido o en otro. Únicamente habrá que conectar la salida del sensor a un pin de entrada digital en la GPIO de la Raspberry Pi.

A continuación se detalla mediante imágenes el montaje paulatino del circuito de lectura de los encoders en una pequeña protoboard:

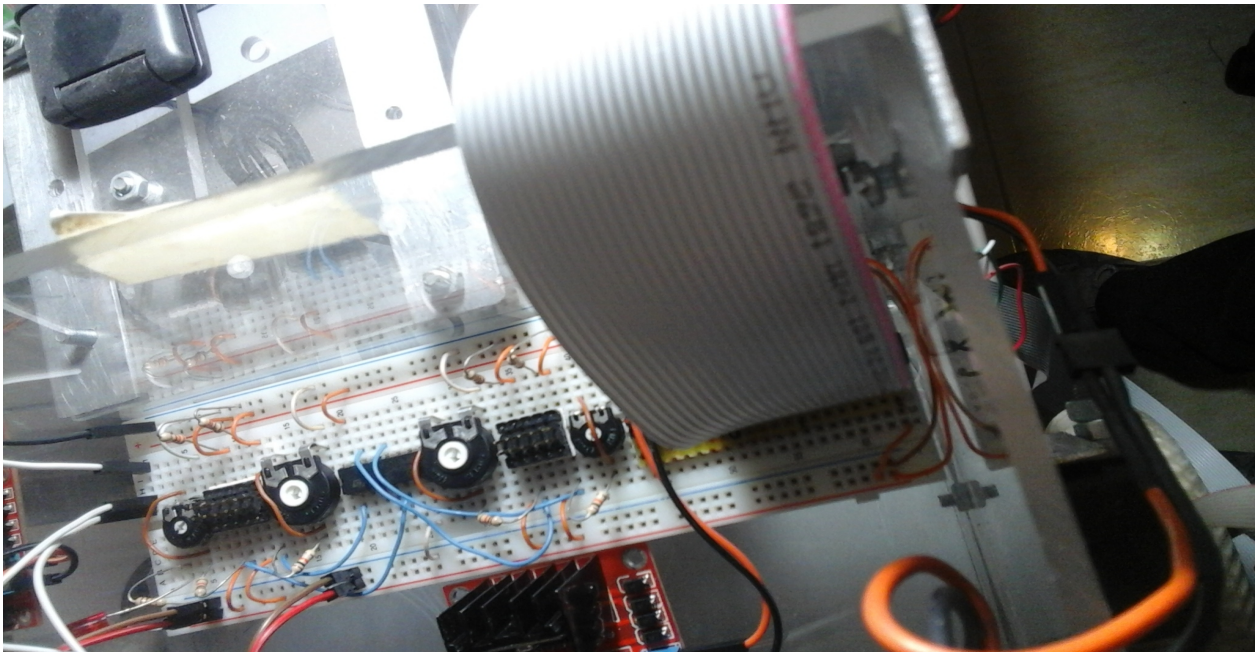


Ilustración 47: Montaje inicial circuito lectura encoders en el robot

Fase inicial, primeros potenciómetros del circuito, conectores donde irán insertados los conectores de los sensores para el giro de las ruedas y en el centro el buffer trigger (HCF40106BE).

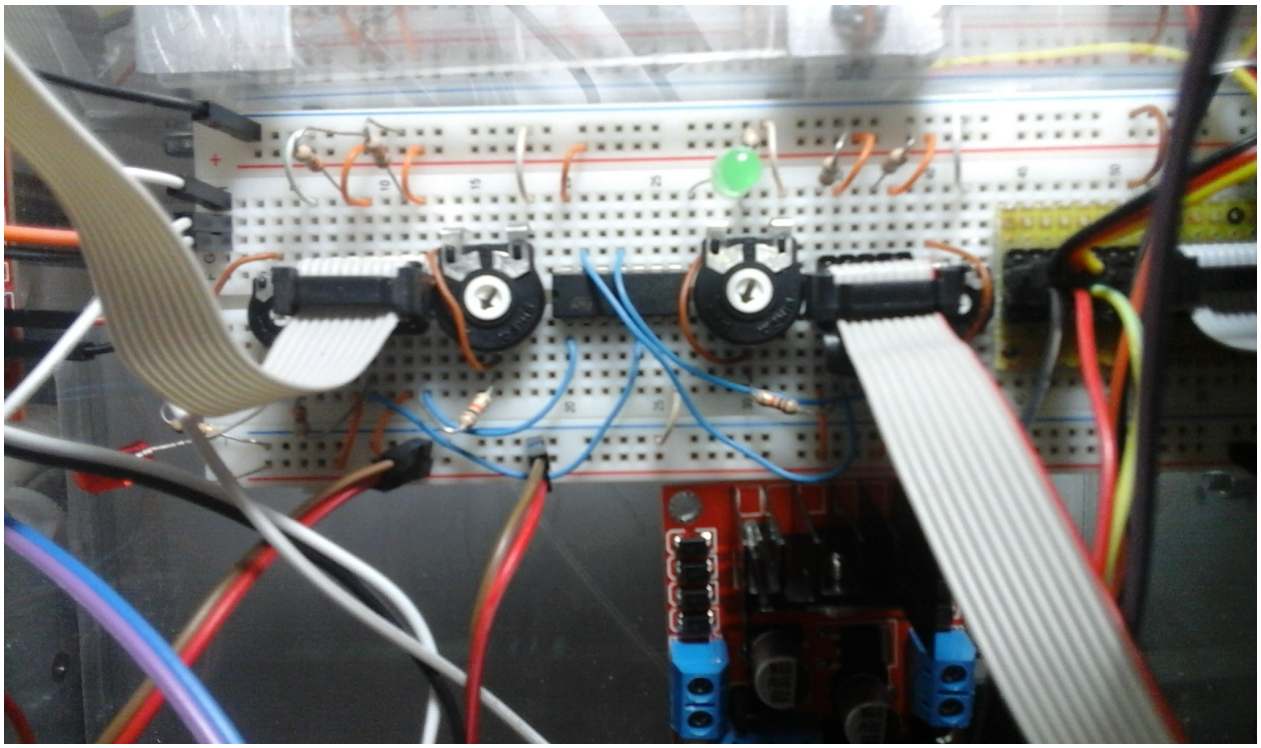


Ilustración 48: Montaje del Circuito Lectura de Encoders

Fase intermedia, con los conectores de los sensores de las ruedas conectados, potenciómetros, resistencias y los dos primeros diodos LED, rojo para el encendido del circuito indicando que está encendido y verde para el resultado de la lectura de los encoders, de hecho, los diodos LED verdes deben utilizarse a pares puesto que trabajaremos con dos señales desfasadas 90°.

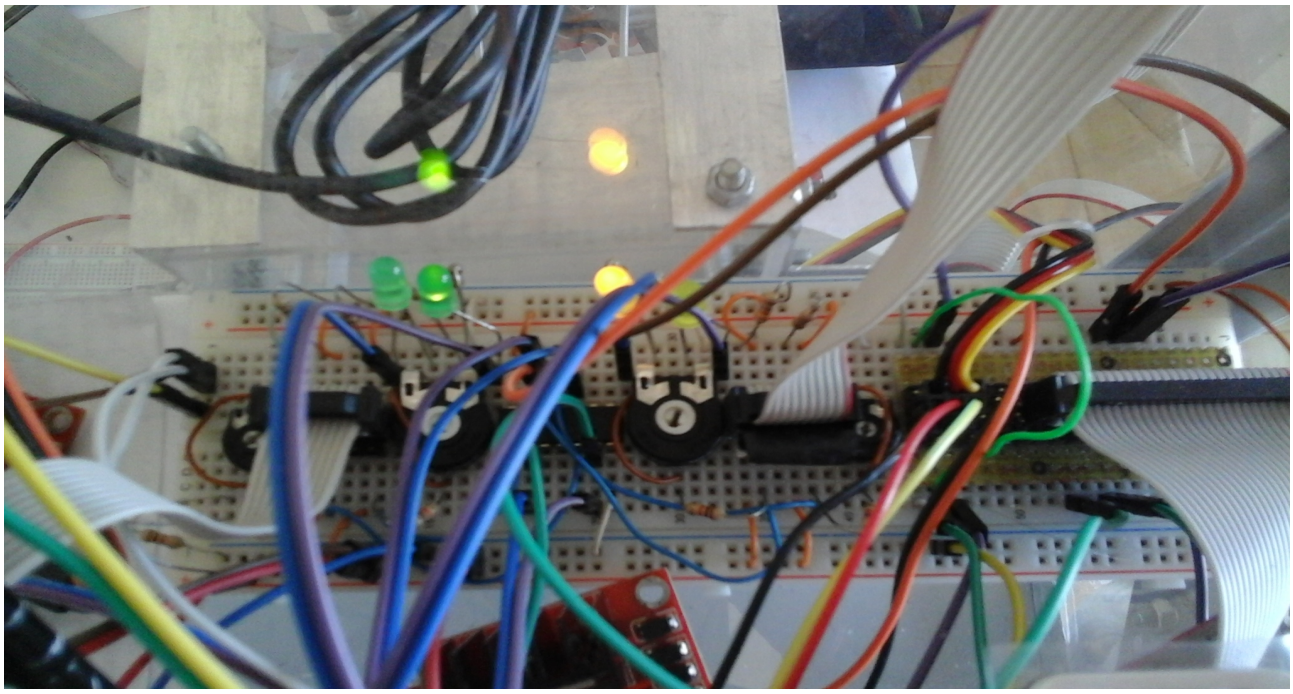


Ilustración 49: Montaje final circuito lectura encoders

Una vez montado el circuito como indicamos en la imagen anterior se procede a la detección del sentido del giro utilizando un osciloscopio en el laboratorio. Es importante localizar un patrón de giro que nos indique si el desplazamiento de los motores se produce hacia adelante o bien hacia detrás.

Conectando el osciloscopio digital a las salidas de los encoders podremos observar el comportamiento de las señales:

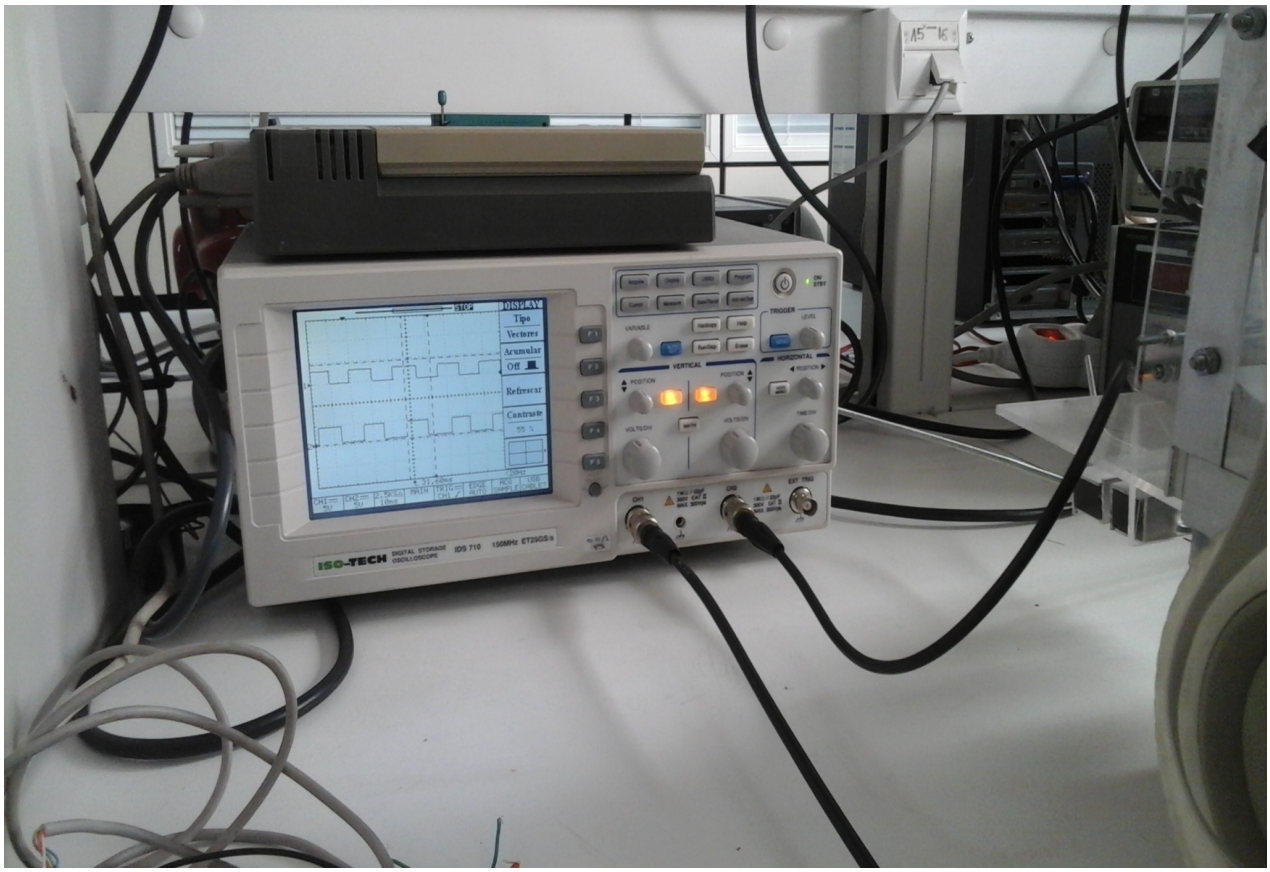


Ilustración 50: Visualización señal encoders con osciloscopio en laboratorio

Fijándonos mejor en el resultado del osciloscopio podremos detectar un cierto patrón con el que descubriremos que los estados con valores altos y valores bajos, es decir, 1's y 0's, se repiten y por lo tanto, los valores son cíclicos:

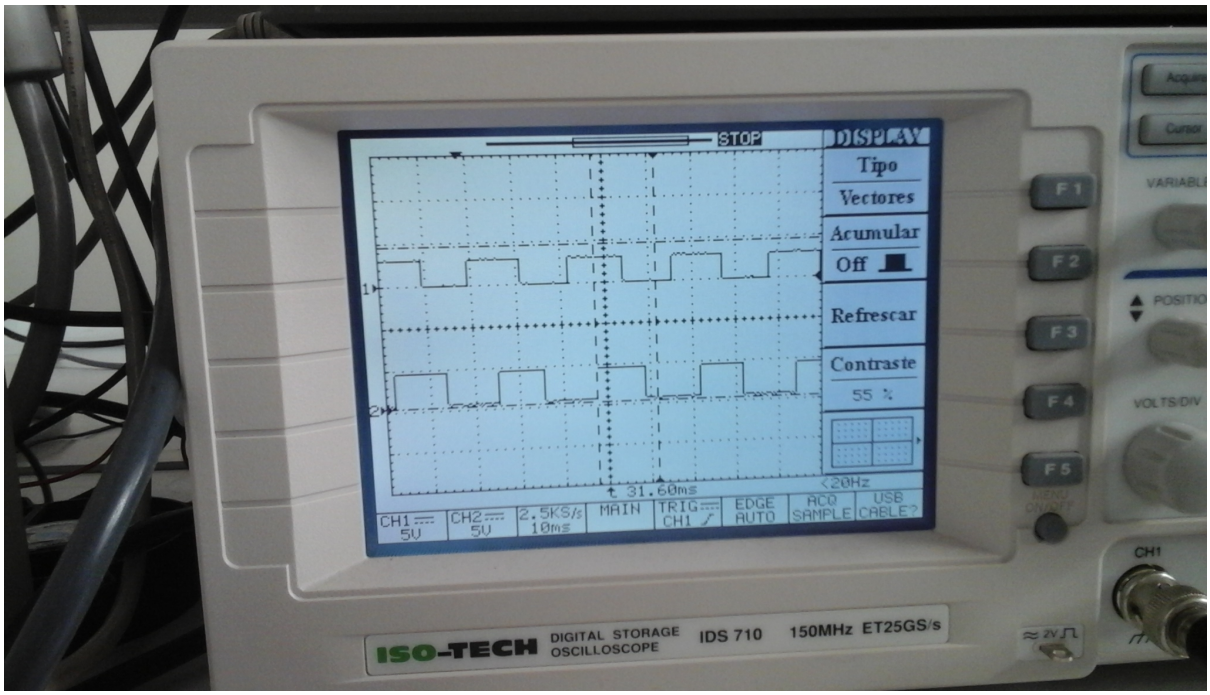


Ilustración 51: Señal lectura encoders en osciloscopio

Para poder ver mejor ese patrón vamos ampliar la imagen obtenida en el osciloscopio y situar lineas verticales sobre la imagen que nos ayuden a identificarlo mejor:

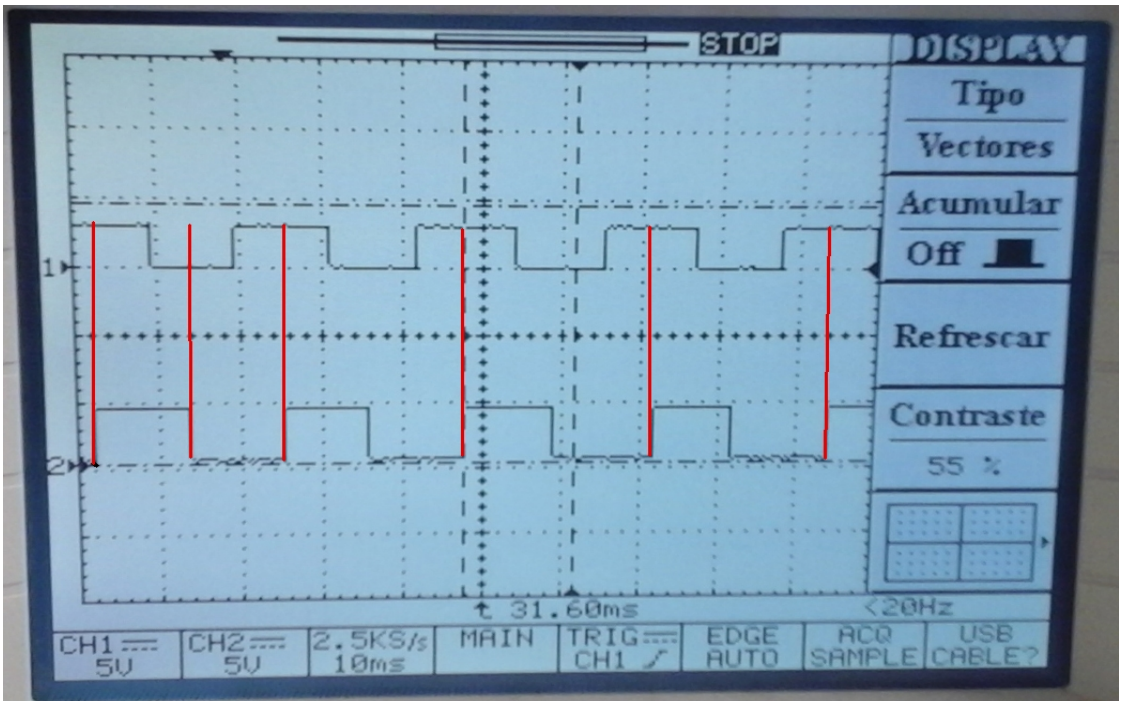


Ilustración 52: señal encoders en osciloscopio

Una vez ampliada la imagen, marcamos los valores altos y bajos (bits) y podemos obtener lo siguiente:

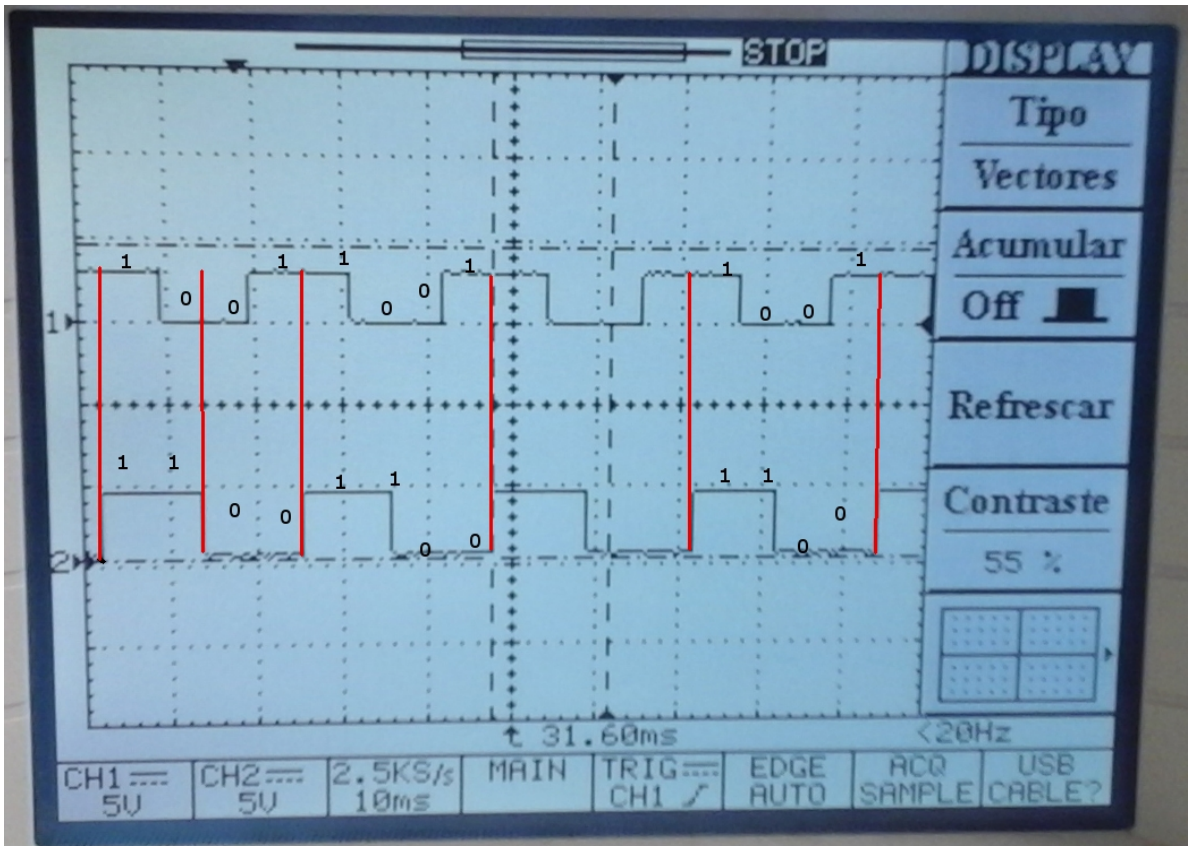


Ilustración 53: Patrón en Señal lectura encoders en osciloscopio

El movimiento de uno de los motores en sentido hacia adelante ha obtenido la siguiente codificación de bits de información en la pantalla del osciloscopio:

CanalA: 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1

CanalB: 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0

Por tanto, ahí ya tenemos la secuencias de bits en la lectura de los encoders, el patrón se repite continuamente entonces, sabemos que si el robot móvil a la lectura de sus encoders están en una posición:

Canal A: 0
Canal B: 0

y

recibe en dicha lectura:

Canal A: 1
Canal B: 0

sabe que se está desplazando hacia adelante y si por el contrario, para la misma lectura de antes (0,0) recibiera:

CanalA: 0
CanalB: 1

sabría entonces que se está desplazando hacia detrás.

Será esta, pues, la referencia que ayudará al robot a conocer hacia donde se desplaza y por cierto, a cada uno de estos cambios de bits 11 a 10 ó 01, 10 a 00 ó 10, 00 a 01 ó 10 y 01 a 11 ó 00 los llamaremos **"Ticks"**.

A partir de ahora, el robot entenderá para saber cuánto tiene que desplazarse en el sentido que se le indique, un número determinado de ticks que cubrirán las distancias que precise moverse.

Con la codificación obtenida vamos a formar un pequeño diagrama de estados que nos indicarán "hacia donde vamos" en cada momento y "de dónde venimos". Sabremos que si por cualquier motivo, el robot ha perdido "la cuenta" de ticks porque por ejemplo, ha colisionado con un objeto que le haya hecho perder dicha "cuenta", a través del siguiente diagrama sabrá en "qué lugar" se encuentra y por dónde lleva exactamente dicha "cuenta" sin perder ticks por el camino:

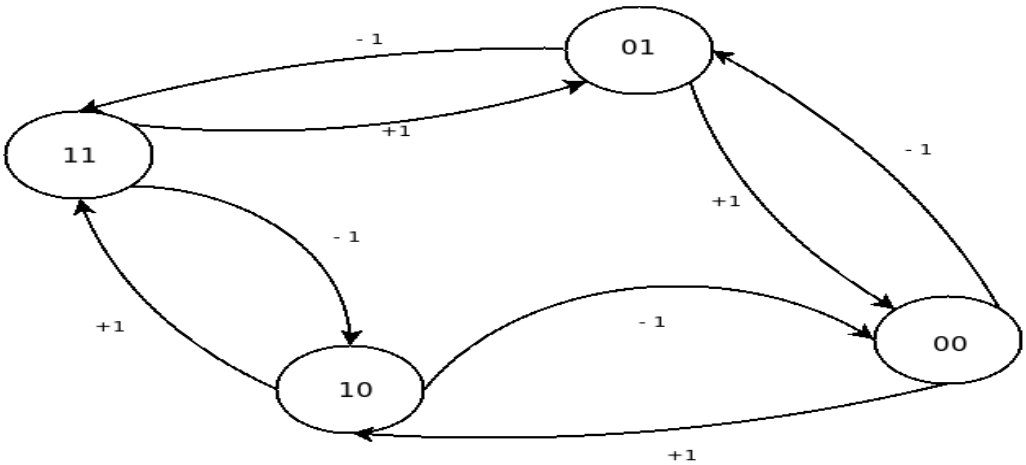


Ilustración 54: Diagrama de estados para lectura encoders

El conteo de los números positivos significa que el desplazamiento del robot se produce hacia adelante y el conteo de los números negativos justo al contrario, el robot se desplaza hacia detrás.

Implementar el diagrama de estados en el código de Python dentro de una función quedaría de la siguiente forma:

```
global contador1
global contador2

anterior_izq_A=RPIO.input(17)
anterior_izq_B=RPIO.input(22)

anterior_der_A=RPIO.input(24)
anterior_der_B=RPIO.input(25)

while running:

        input_value_17 = RPIO.input(17)
        input_value_22 = RPIO.input(22)


        if (input_value_17 != anterior_izq_A or input_value_22 != anterior_izq_B):
                #print " leyendo encoder adelante"
                # IZQ = 0  DER = 0
                if (anterior_izq_A == False and anterior_izq_B == False):
                        # SIG_IZQ = 1 SIG_DER = 0
                        if (input_value_17 == True and input_value_22 == False):
                                contador1 = contador1 + 1

                        # SIG_IZQ = 1 SIG_DER = 1
                        elif (input_value_17 == True and input_value_22 == True):
                                contador1 = contador1 + 2

                        # SIG_IZQ = 0 SIG_DER = 1
                        elif (input_value_17 == False and input_value_22 == True):
                                contador1 = contador1 + 3
```

Ilustración 55: Código de Python del diagrama de estados

Utilizo los contadores globales para contar el número de ticks, uno para cada motor, lectura de los encoders correspondientes, 2 para cada motor. El motor izquierdo utiliza los pines 17 y 22 en la GPIO y el motor derecho el 24 y el 25. Al contar con dos encoders se hace uso de dos pines para cada uno. Inicializamos dos variables para los estados anteriores de los dos encoders y dentro de un bucle controlamos las posibles combinaciones en cada estado (00, 01, 10, 11) con las combinaciones que nos van llegando en la siguientes lecturas. Actualizamos las variables y continuamos la lectura.

```
            # IZQ = 0    DER = 1
            elif (anterior_izq_A== False and anterior_izq_B == True):

                    # SIG_IZQ = 0 SIG_DER = 0
                    if (input_value_17 == False and input_value_22 == False):
                            contador1 = contador1 + 1

                    # SIG_IZQ = 1 SIG_DER = 0
                    elif (input_value_17 == True and input_value_22 == False):
                            contador1 = contador1 + 2

                    # SIG_IZQ = 1 SIG_DER = 1
                    elif (input_value_17 == True and input_value_22 == True):
                            contador1 = contador1 + 3

            # IZQ = 1    DER = 0
            elif (anterior_izq_A== True and anterior_izq_B == False):

                    # SIG_IZQ = 1 SIG_DER = 1
                    if (input_value_17 == True and input_value_22 == True):
                            contador1 = contador1 + 1

                    # SIG_IZQ = 0 SIG_DER = 1
                    elif (input_value_17 == False  and input_value_22 == True):
                            contador1 = contador1 + 2

                    # SIG_IZQ = 0 SIG_DER = 0
                    elif (input_value_17 == False and input_value_22 == False):
                            contador1 = contador1 + 3

            # IZQ = 1    DER = 1
            elif (anterior_izq_A== True and anterior_izq_B == True):

                    # SIG_IZQ = 0 SIG_DER = 1
                    if (input_value_17 == False and input_value_22 == True):
                            contador1 = contador1 + 1

                    # SIG_IZQ = 0 SIG_DER = 0
                    elif (input_value_17 == False  and input_value_22 == False):
                            contador1 = contador1 + 2

                    # SIG_IZQ = 1 SIG_DER = 0
                    elif (input_value_17 == True and input_value_22 == False):
                            contador1 = contador1 + 3

    anterior_izq_A = input_value_17
    anterior_izq_B = input_value_22
```

Ilustración 56: Código Python para diagrama estados y cuenta de Ticks Rueda Izquierda

Para el motor derecho el código es el mismo pero usando su contador correspondiente y sus encoders que estarán conectados a distintos pines de GPIO en la Raspberry Pi:

```
# MOTOR DERECHO

input_value_24 = RPIO.input(24)
input_value_25 = RPIO.input(25)

if (input_value_24 != anterior_der_A or input_value_25 != anterior_der_B):
        #  IZQ = 0  DER = 0
        if (anterior_der_A == False and anterior_der_B == False):

                # SIG_IZQ = 1 SIG_DER = 0
                if (input_value_24 == True and input_value_25 == False):
                        contador2 = contador2 + 1

                # SIG_IZQ = 1 SIG_DER = 1
                elif (input_value_24 == True and input_value_25 == True):
                        contador2 = contador2 + 2

                # SIG_IZQ = 0 SIG_DER = 1
                elif (input_value_24 == False and input_value_25 == True):
                        contador2 = contador2 + 3

        # IZQ = 0    DER = 1
        elif (anterior_der_A== False and anterior_der_B == True):

                # SIG_IZQ = 0 SIG_DER = 1
                if (input_value_24 == False and input_value_25 == False):
                        contador2 = contador2 + 1

                # SIG_IZQ = 1 SIG_DER = 0
                elif (input_value_24 == True and input_value_25 == False):
                        contador2 = contador2 + 2

                # SIG_IZQ = 1 SIG_DER = 1
                elif (input_value_24 == True and input_value_25 == True):
                        contador2 = contador2 + 3

        # IZQ = 1    DER = 0
        elif (anterior_der_A== True and anterior_der_B == False):

                # SIG_IZQ = 1 SIG_DER = 1
                if (input_value_24 == True and input_value_25 == True):
                        contador2 = contador2 + 1

                # SIG_IZQ = 0 SIG_DER = 1
                elif (input_value_24 == False  and input_value_25 == True):
                        contador2 = contador2 + 2

                # SIG_IZQ = 0 SIG_DER = 0
                elif (input_value_24 == False and input_value_25 == False):
                        contador2 = contador2 + 3
```

Ilustración 57: Código Python para diagrama estados y cuenta Ticks Rueda derecha (1)

```
                # IZQ = 1    DER = 1
                elif (anterior_der_A== True and anterior_der_B == True):

                        # SIG_IZQ = 0 SIG_DER = 1
                        if (input_value_24 == False and input_value_25 == True):
                                contador2 = contador2 + 1

                        # SIG_IZQ = 0 SIG_DER = 0
                        elif (input_value_24 == False  and input_value_25 == False):
                                contador2 = contador2 + 2

                        # SIG_IZQ = 1 SIG_DER = 0
                        elif (input_value_24 == True and input_value_25 == False):
                                contador2 = contador2 + 3


        anterior_der_A = input_value_24
        anterior_der_B = input_value_25

        #print contador1
        #print contador2
#END_WHILE
hilo.stopped = True
return([contador1, contador2])
```

Ilustración 58: Código Python para diagrama estados y cuenta Ticks Rueda derecha (2)

El código finaliza con la salida del bucle y para el hilo que se encarga de ejecutar esta parte. Devuelve los contadores de ticks al término de su función.

El código podrá visualizarse en su totalidad en los anexos, también más adelante en el apartado de software de este documento, se explicará con más detenimiento.

4.2.3.8. Circuito para la Alimentación.

Es de vital importancia alimentar toda la circuitería correctamente.
En las condiciones iniciales del proyecto se incluía una fuente de alimentación que actuaba como regulador de tensión entre los circuitos del robot y la fuente externa que bien, podría ser una pila de 11.1V o incluso una fuente externa que pudiéramos acoplar para realizar las pertinentes pruebas.

El aspecto de la fuente es el siguiente:

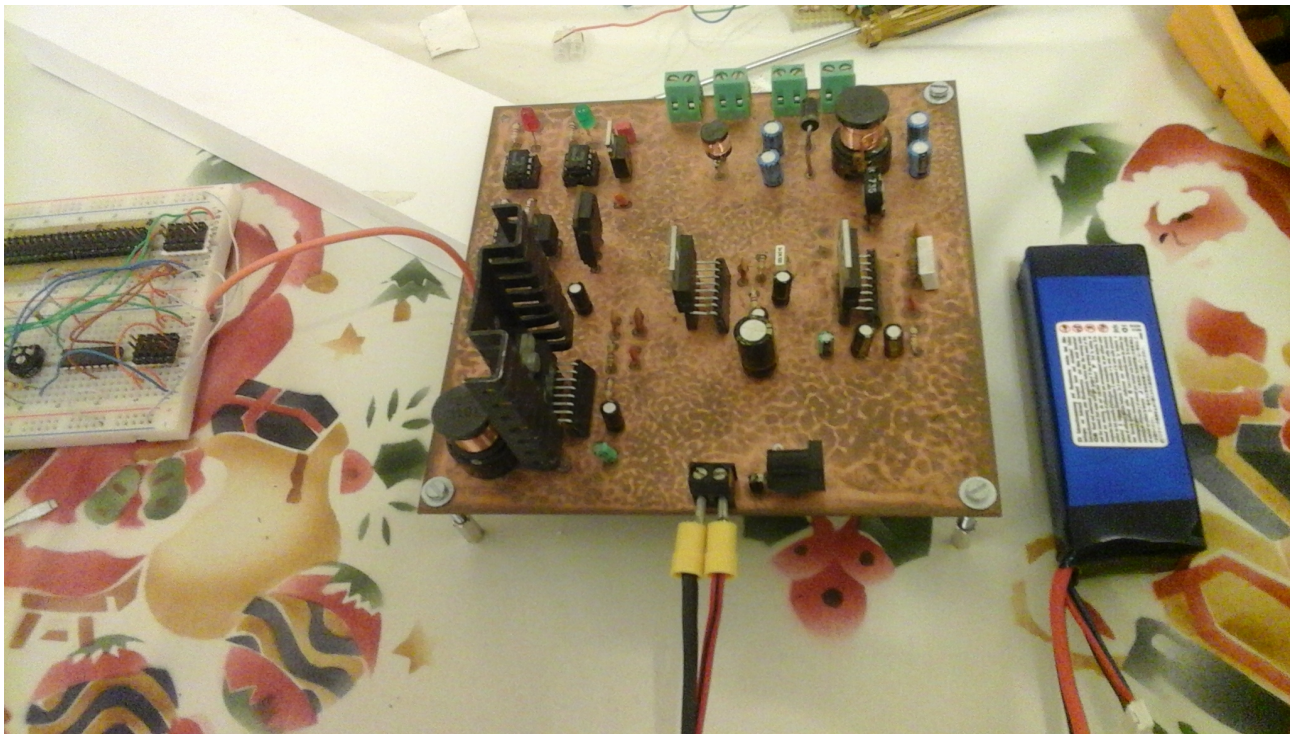


Ilustración 59: Fuente de alimentación inicial del prototipo

Al principio la fuente se utilizó pero llegado su tasa de error y comportamiento y alta sensibilidad al ruido, introdujo fallos por lo que fue sustituida.

Durante el transcurso de este proyecto dicha fuente es posible que entrara en contacto con la humedad en el aire y dejase de funcionar correctamente, también sus componentes estaban poco estables en cuanto a soldadura en la placa.

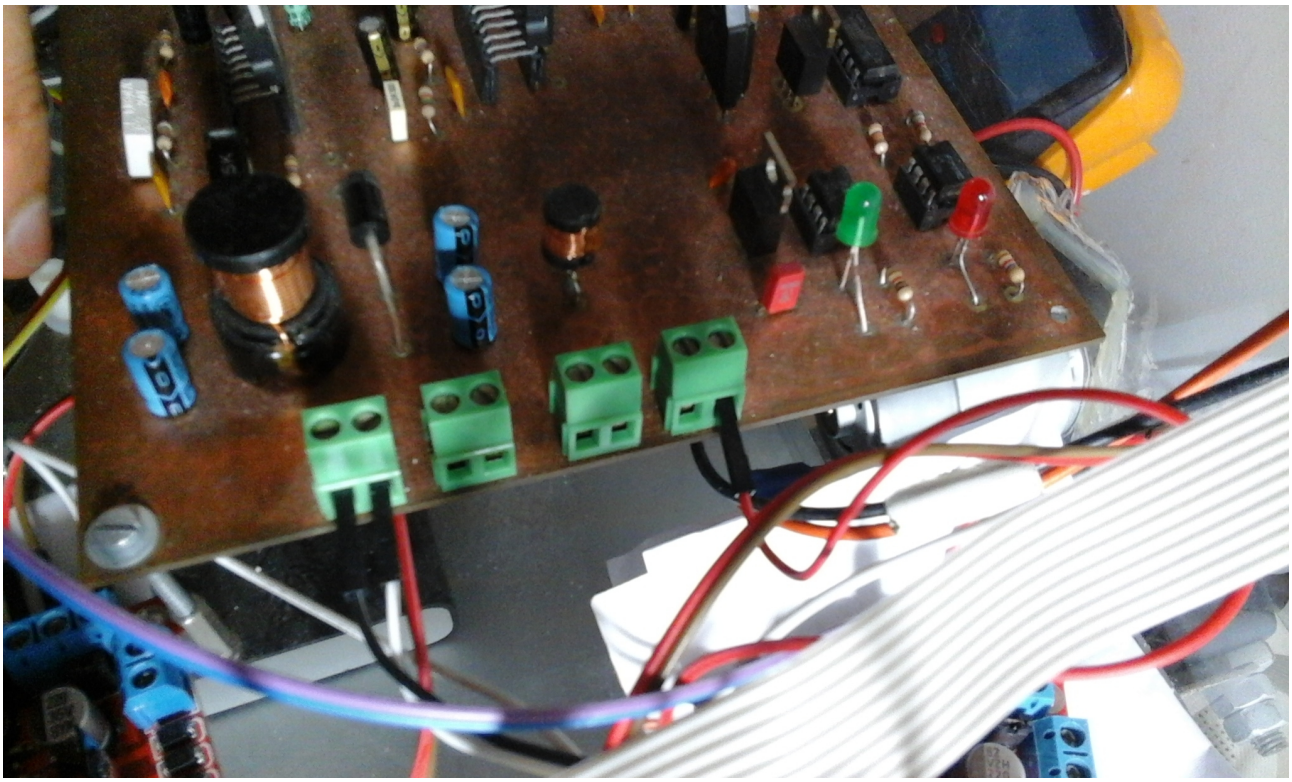


Ilustración 60: Fuente alimentación inical prototipo conectada

En su defecto, tuvo que implementarse un circuito sencillo que hará las veces de regulador de tensión:

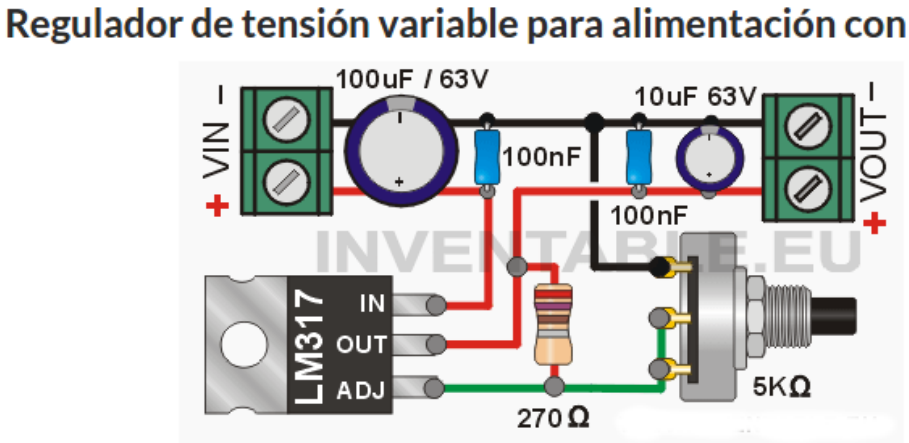


Ilustración 61: Esquema conexiones regulador de tensión

La tensión de entrada puede ser entre 5 Voltios y 30 Voltios, mientras que la tensión a la salida que viene regulada por el potenciómetro, oscila desde 1.25 Voltios hasta 28 Voltios. Los capacitadores son imprescindibles para filtrar los ruidos y evitar fenómenos de autooscilación del regulador de tensión.

El LM317 es un regulador de tensión lineal ajustable capaz de suministrar a su salida en condiciones normales un rango que va desde 1,2 hasta 37V Voltios y una intensidad de 1,5 Amperios. Posee tres patillas que como indican en la figura anterior, son línea de ajuste (ADJ), entrada (IN) y salida (OUT).

Debido a la corriente de salida y a la diferencia entre tensiones que existen a la entrada y salida, es necesario utilizar un disipador para el componente LM317.

Observar en el esquema que el componente electrónico fue preparado para regular la tensión de un dispositivo solamente, por lo que se ha soldado en placa dos componentes según el esquema anterior para controlar la alimentación de los circuitos y la alimentación de los Servos. Un servomotor es un dispositivo similar a un motor de corriente continua que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación y mantenerse estable en dicha posición. Un servo se puede controlar tanto en posición como en velocidad. La intención es provocar una salida de 5 Voltios para alimentar el circuito y otra salida de 7 Voltios para alimentar los servomotores encargados de recoger las pelotas de tenis y para liberarlas una vez colmada su capacidad de recogida.

El Regulador de Tensión quedó soldado como puede apreciarse en la siguientes imágenes:

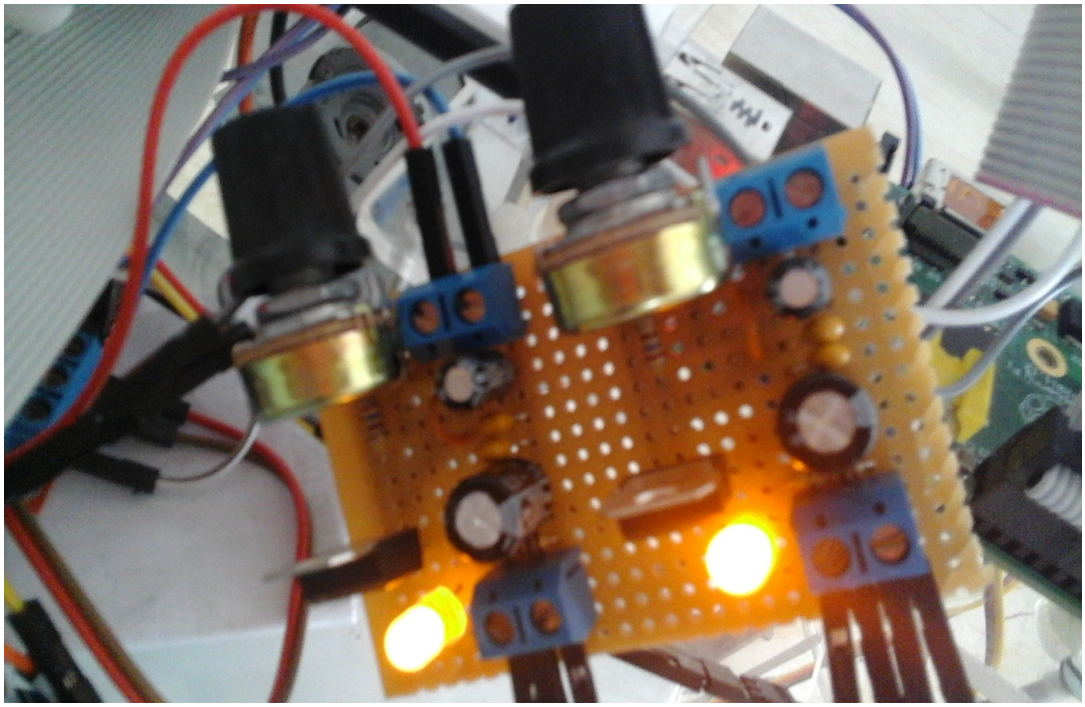


Ilustración 62: Regulador de tensión en funcionamiento

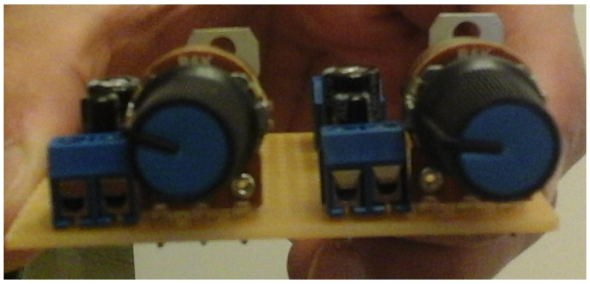


Ilustración 63: Vista frontal de regulador de tensión

4.2.3.9. Alimentación para los servomotores.

En la circuitería del apartado anterior para la alimentación hemos observado que se ha quedado libre una segunda salida y que se estaban utilizando dos reguladores como se indicó. La idea era que uno de los reguladores de tensión pudiera situarse en la medida de 5 Voltios para los motores y otro, en la medida de 7 Voltios para los servos. Los servos del robot móvil están diseñados para recoger las pelotas dentro del habitáculo habilitado para ello y para que, una vez recogidas las pelotas, no se caigan del robot sino cuando el prototipo juzgue necesario vaciar el vehículo en "la estación de entrega de pelotas de tenis".

El aspecto que tiene el servomotor delantero para recoger las pelotas es el siguiente:

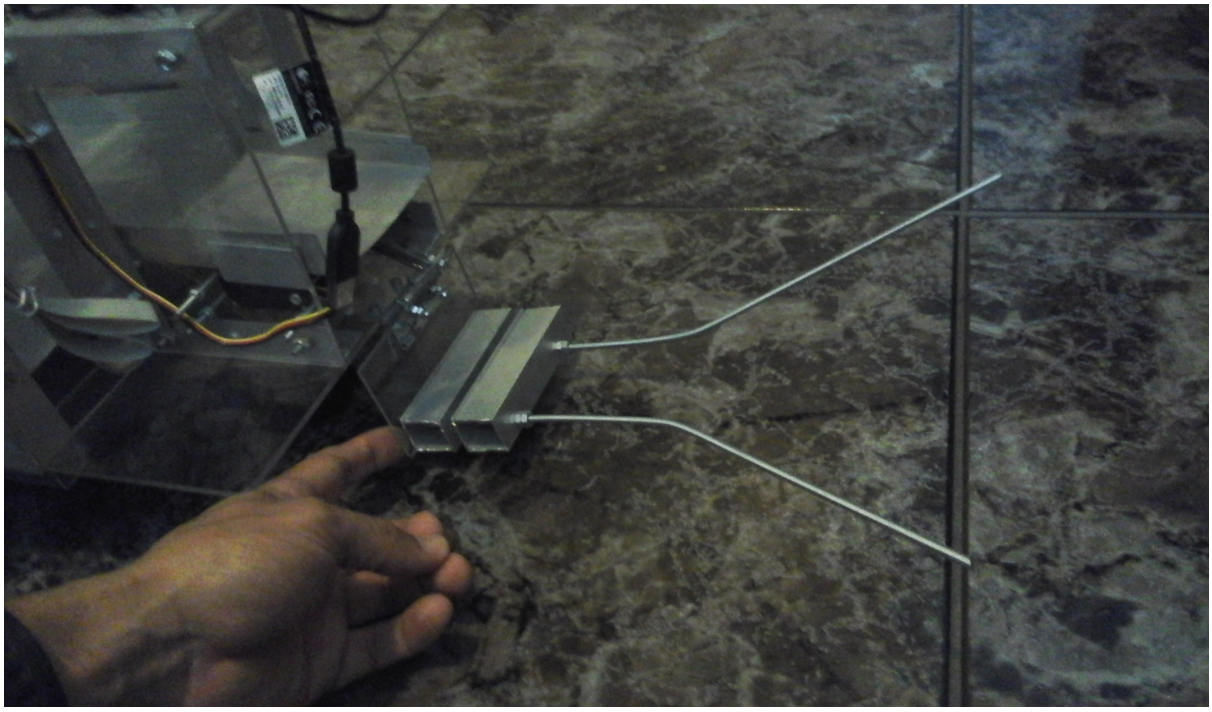


Ilustración 64: Vista Lateral mecanismo delantero recogida pelotas

El funcionamiento es básico, el robot se desplaza hasta situar la pelota de tenis entre los alambres de su antena y cuando está preparado el servomotor levanta una cierta cantidad de grados para que, mecánicamente, introduzca la pelota en la cavidad superior, almacenándola en su interior. El servomotor se encuentra en la parte delantera, tal y como puede verse en la siguiente imagen:

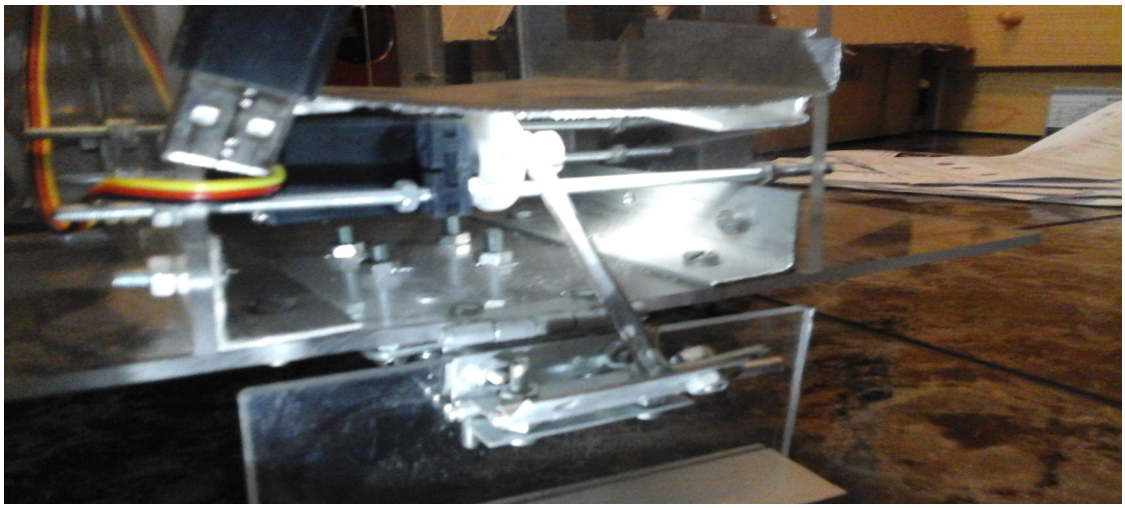


Ilustración 65: Vista frontal servo delantero

Para que las pelotas de tenis recogidas no se salgan del interior del robot, existe un servomotor en la parte trasera que baja y sube una pieza metálica en forma de “L”:

Ilustración 66: Servo y mecanismo trasero

El funcionamiento del servomotor es el mismo que el servo de recogida delantero. Cuando estima oportuno vaciar el interior de la caja de metacrilato del robot, levanta la pieza metálica y las pelotas caen de la bandeja inclinada de su interior:

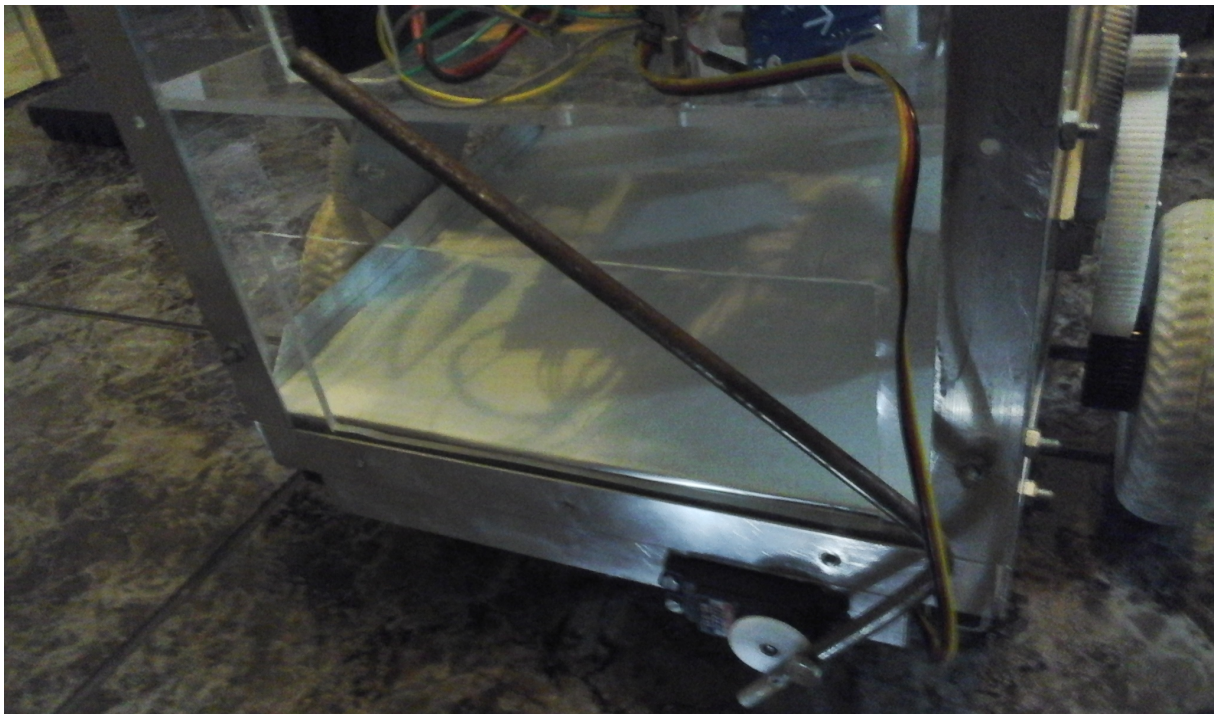


Ilustración 67: Servo y mecanismo trasero en funcionamiento

Existe un problema importante a la hora de utilizar los dos servos y es su alimentación. Si utilizamos el circuito anteriormente descrito es muy posible que se "sobrecaliente" al uso de ambos servomotores en el prototipo, es decir, el circuito de regulador de tensión del apartado anterior no dispone de un buen disipador de calor y el uso de estos servos adicionales podrían hacer que la temperatura de la placa donde están soldados los componentes, exceda de valores que pueda resistir por lo que se ha tomado la decisión de utilizar un elemento electrónico que nos ayude a superar ese pequeño contratiempo y limitación.

El elemento electrónico utilizado para ello será el **Módulo LM2596.** Se trata de un convertidor de Voltaje DC-DC Buck 1.25V-35V. Este dispositivo electrónico, como su nombre indica, ya viene montado en una placa y por tanto apenas necesita algún tipo de modificación (que en realidad si la hubo) para adaptarse al prototipo que he desarrollado en este proyecto.

El aspecto del módulo es el siguiente:

Ilustración 68: Vista superior modulo LM2596

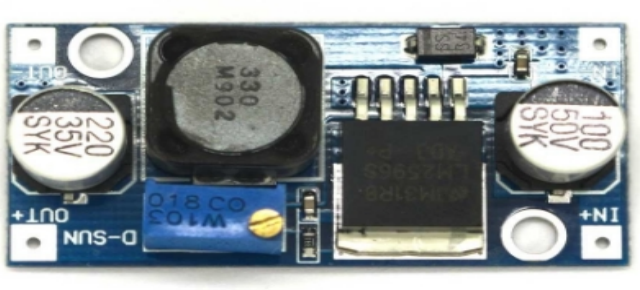


Ilustración 69: Vista superior modulo LM2596 (2)

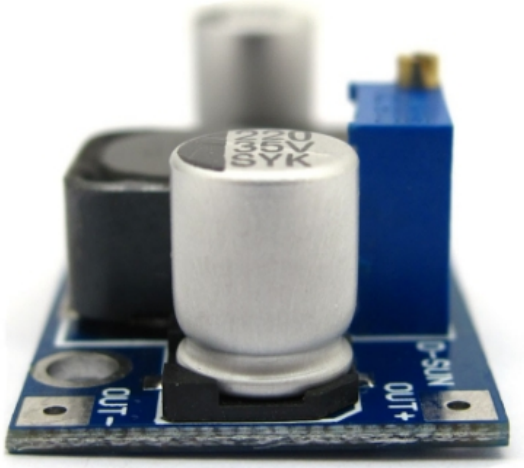


Ilustración 71: Vista lateral salidas LM2596

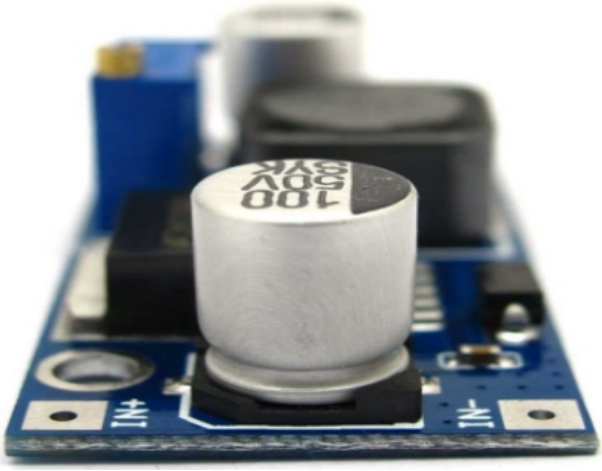


Ilustración 70: Vista lateral entradas LM2596

Este módulo esta basado en el Regulador DC-DC Step Down LM2596 que es un circuito integrado monolítico adecuado para el diseño fácil y conveniente de una fuente de conmutación tipo buck. Es capaz de conducir una corriente de hasta 3A. Maneja una carga con excelente regulación de línea y bajo voltaje de rizado. Este dispositivo está disponible con voltaje de salida ajustable. El módulo reduce al mínimo el uso de componentes externos para simplificar el diseño de fuentes de alimentación. Es decir, te permite tener un voltaje regulado a partir de una fuente de alimentación con un voltaje mayor, por ejemplo si tienes una fuente de 12V puedes regularlos a 5V, 3.3V, 2.2V, etc, para el uso con microcontroladores, Arduino, PICs, Raspberry Pi, fuentes variables, drivers para leds, etc.

El módulo convertidor LM2596 es una fuente de alimentación conmutada, así que su eficiencia es significativamente mayor en comparación con los populares reguladores lineales de tres terminales, especialmente con tensiones de entrada superiores.

Sus características se resumen en:

- Basada en el regulador LM2596, salida entre 1,5 y 35Vdc
- Voltaje de entrada: 4.5-40V
- Voltaje de salida: 1.5-35V (Adjustable)
- Corriente de salida: Maxima 3A
- Dimensiones: 43*20*14mm
- Frecuencia de switching: 150 KHz

Nótese en las imágenes que, el módulo posee dos entradas (-IN e +IN) y dos salidas (-OUT y +OUT) en unos agujeros sobre unos zócalos plateados donde debemos conectar los extremos de los cables para la alimentación de los servomotores.

Se ha procedido a soldar sobre las entradas y salidas del módulo LM2596 unos conectores tipo monopines (headers) macho ambos extremos que facilitarán las conexiones de los cables tipo monopin que estamos usando para las conexiones de todos los componentes del prototipo.

El aspecto de dichos conectores soldados al módulo es el siguiente:

Ilustración 72: Conectores Monopines Headers

Los conectores vienen agrupados pero se pueden cortar fácilmente con una herramienta tipo “corta-rentes”.

Tras haberlos recortado y soldado, el nuevo aspecto del módulo es el siguiente:

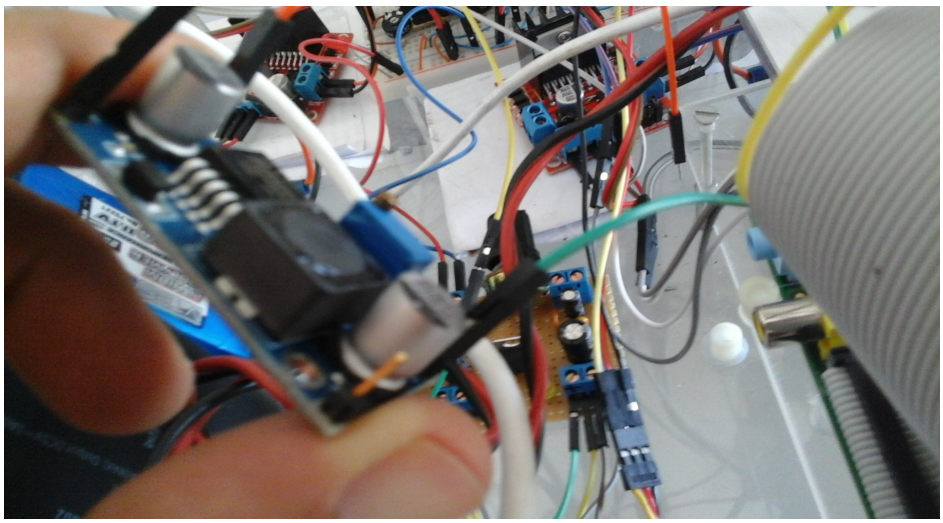


Ilustración 73: Visión modulo LM2596 conectado

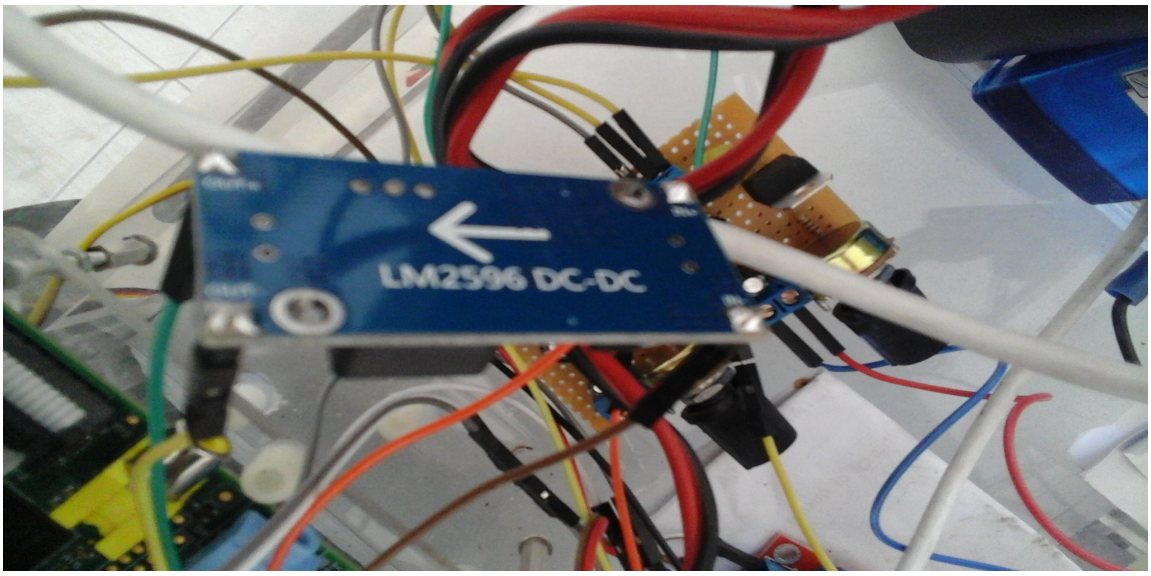


Ilustración 74: Visión modulo LM2596 parte trasera conectado

Esto nos ayudará bastante con las conexiones y facilitará el trabajo de conectarlo con la protoboard y alimentar los servo-motores.

4.2.3.10. Cálculos para desplazamiento del robot móvil.

Como las matemáticas indican, la circunsferencia de un círculo se puede definir como π * D, dónde π ó "pi" es una constante en torno a 3.14 y D es el diámetro del círculo.

Desde estas reglas podemos dirigir el robot móvil sobre una distancia específica sin pruebas ni errores, sólo conociendo el diámetro de la rueda.

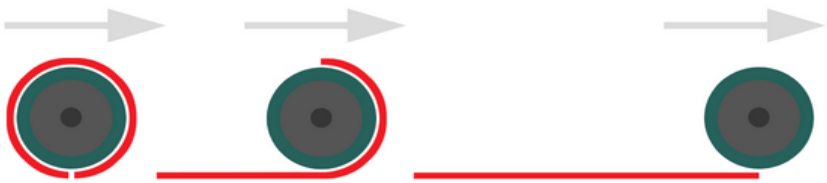


Ilustración 75: Recorrido lineal en el giro de una rueda

Es decir, conociendo el diámetro de la rueda del robot que son 11 centímetros podremos saber que, cuando la rueda a girado una vez por completo, ha recorrido una distancia dada por la fórmula:

$\pi * D = 3.141592654 \times 11 = 34{,}557519189 \approx 34.5575$ centímetros.

Aprovechando el contador de ticks que presentamos en el código del programa hecho en Python y de forma manual, haciendo una marca en la rueda de principio-fin (para controlar el giro completo de la rueda) para luego girar la rueda con la mano, obtendremos el número de ticks que se necesitan para dar una vuelta completa.

Por motivos de mecánica de los motores en uno de ellos hemos tomado una medida un poco distinta que el otro:

- Para el motor izquierdo hemos obtenido un total de ticks para el giro completo de una rueda. Es decir, el motor izquierdo emplea ticks para desplazarse 34.5575 centímetros tanto hacia adelante como hacia atrás.

- Para el motor derecho hemos obtenido un total de ticks para desplazar 34.5575 el robot.

A través de una sencilla regla de tres obtenemos lo siguiente:

Motor Izquierdo: 86 - 34.5575
1 - X 1 tick = 0.40183 cms

Motor Derecho: 89 - 34.5575
1 - X 1 tick = 0.38828 cms

Estos valores son simplemente para aclarar lo que corresponde un tick en centímetros ya que para el código nos bastará con utilizar el número de ticks por vuelta que efectúa cada rueda tal y como presenta la siguiente imagen de la captura del código de la aplicación:

```
global PI
PI = 3.14

global dos_pi_R
dos_pi_R = 34.5575

global TOTAL_TICKS_VUELTA_IZQ
TOTAL_TICKS_VUELTA_IZQ = 86
global TOTAL_TICKS_VUELTA_DER
TOTAL_TICKS_VUELTA_DER = 89
```

Ilustración 76: Código declaración variables globales con valores de ticks para cada motor

Aplicando la fórmula:

Num ticks motor derecho = (TOTAL_TICKS_VUELTA_DER * centimetros) / π * D

Num ticks motor izquierdo = ((TOTAL_TICKS_VUELTA_IZQ + tolerancia) * centimetros) / π * D

dónde centímetros es la distancia en centímetros que le indicaremos al robot que deseamos desplazarnos. Este desplazamiento viene en función de las distancias que habrán respecto al objetivo que no es otro que el de recoger las pelotas de tenis de la pista.

```
# NUMERO DE TICKS.
centimetros = int(sys.argv[2])
if (sentido == "ADELANTE") or (sentido == "ATRAS"):
        num_ticks_der = (TOTAL_TICKS_VUELTA_DER * centimetros)/dos_pi_R
        num_ticks_izq = ((TOTAL_TICKS_VUELTA_IZQ + tolerancia) * centimetros)/dos_pi_R
```

Ilustración 77: Codigo cálculo ticks en el desplazamiento

La tolerancia es un factor que aplicamos para intentar reducir la diferencia de ticks que puede realizar un motor respecto al otro ya que por motivos de mecánica existe un motor que gira “mejor” que el otro. Tolerancia vale 3 para que el motor izquierdo haga todo lo posible por girar como lo hace el motor derecho.

Cuando el robot tiene que realizar un giro podemos analizar lo siguiente:

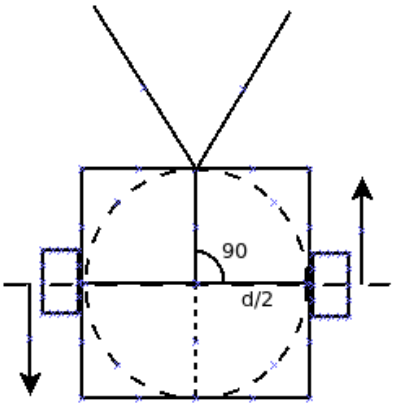


Ilustración 78: Representación del Giro del robot

Podemos ver en este diagrama que el giro será hacia la izquierda. Tenemos el radio de la circunsferencia (la mitad del diámetro) y cuatro cuadrantes.

Esos cuatro cuadrantes podemos dividirlos en grados tal y como nos indica la siguiente imagen:

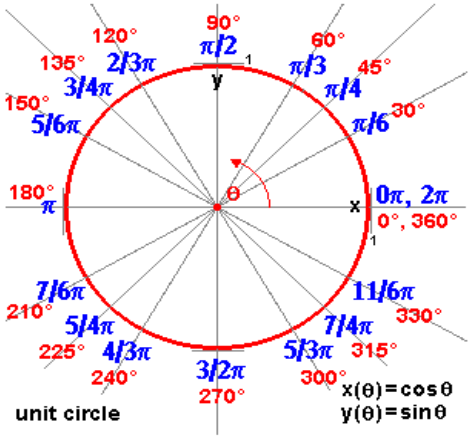


Ilustración 79: Dibujo de ángulos en grados y radianes

Utilizando la ecuación

$$\frac{\pi * d}{4} = x * 2\pi R$$

eliminando π en la ecuación tenemos que

$$x = \frac{d}{8R}$$

por tanto, para dar 90º hemos tenido que dar (17,2787 / 8 x 5,5) = 0,3927 vueltas.

Entonces, para saber el número de ticks que son ese número de vueltas, debemos utilizar cada rueda por separado:

1 vuelta ---------- 86 ticks (izquierda)

0,3927 ---------- x_ticks_izquierda

x_ticks_izquierda = 33'7720 para girar 90º

1 vuelta ---------- 89 ticks (derecha)

0,3927 ---------- x_ticks_derecha = 34'9501 para girar 90º

Ahora para saber cuántos ticks es un grado dividimos las cantidades:

33'7720 / 90 = 0,3752 ticks para girar un grado la rueda izquierda

34,9501 / 90 = 0,3883 ticks para girar un grado la rueda derecha

Recordemos que estamos utilizando la semidistancia, por lo que **faltaría un factor 2** que haría posicionar el robot en el número exacto de ticks que necesitamos para que gire los grados que le digamos.

4.2.3.11. Movimiento del robot: Sistema de Control.

El robot debe comportarse de la forma más precisa posible en su movimiento, atendiendo a las medidas exactas para poder recoger los objetos que se le marcan como objetivos: las pelotas de tenis. El comportamiento del prototipo precisa en todo momento estar dirigido por un sistema de control.

Un sistema de control es un conjunto de dispositivos encargados de administrar, ordenar, dirigir o regular el comportamiento de otro sistema, con el fin de reducir las probabilidades de fallo y obtener los resultados deseados. Normalmente, estos sistemas de control son utilizados en procesos de producción industriales para controlar diferentes equipos y máquinas.

Un sistema de control debe conseguir los siguientes objetivos:

1) Ser estables y robustos frente a perturbaciones y errores en los modelos.

2) Ser eficientes según un criterio preestablecido evitando comportamientos bruscos e irreales.

Hay muchas maneras de clasificar los sistemas de control pero para resumir y centrarnos en lo que concierne a este proyecto, para nosotros existirán básicamente dos clases de sistemas de control: sistemas de lazo abierto y sistemas de lazo cerrado. La diferencia entre ambos es que, en los sistemas de control de lazo abierto la salida se genera dependiendo de la entrada; mientras que en los sistemas de control de lazo cerrado la salida depende de las consideraciones y corrección realizadas por la retroalimentación. Estos sistemas de control también son conocidos como sistemas de control con realimentación.

Sistemas de control de **lazo abierto**.

Aquel sistema en que solo actúa el proceso sobre la señal de entrada y da como resultado una señal de salida independiente a la señal de entrada pero basada en la primera. Esto significa que no hay retroalimentación hacia el controlador para que este pueda ajustar la acción de control, es decir, la señal de salida no se convierte en señal de entrada al controlador.

Un ejemplo de este sistema de control podría ser el llenado de un tanque a través de una manguera, mientras la manguera siga abierta el agua se vierte y la altura del tanque asciende. No existe manera de controlar si el

vertido para hacer que se cierre la llave a una determinada altura. También un microondas es un sistema de control de lazo abierto donde indicas las temperatura a calentar y el tiempo en hacerlo.

Un sistema de control lazo abierto se caracteriza por:

- Ser sencillos y de fácil concepto.

- Nada asegura su estabilidad si existe perturbación.

- La salida no se compara con la entrada.

- Ser afectado por las perturbaciones que pueden ser tangibles e intangibles.

- La precisión depende de la previa calibración del sistema.

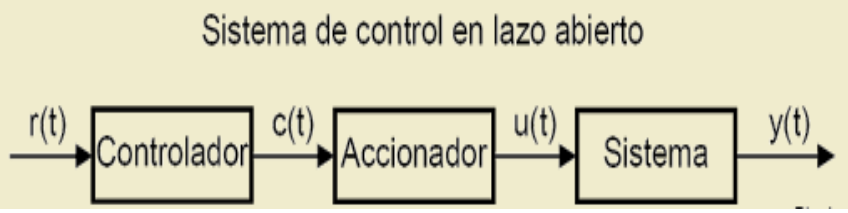


Ilustración 80: Esquema de sistema de control lazo abierto

Dónde

r(t) Referencia. Es el estado que se desea alcanzar en el sistema.
c(t) Control. Es la señal que genera el controlador.
u(t) Accionamiento.
y(t) Salida del Sistema. Estado real que ha alcanzado el sistema a controlar.

Sistemas de control de **lazo cerrado**.

Son los sistemas de control en los que la acción de control está en función de la señal de salida. Estos sistemas usan la retroalimentación desde un resultado final para ajustar la acción de control en consecuencia.

Estos sistemas se utilizan cuando se dan alguna de las siguientes circunstancias:

- Cuando un proceso no es posible de regular por el hombre.

- Una producción a gran escala que exige grandes instalaciones y el hombre no es capaz de manejar.

- Vigilar un proceso de manera continua en la que dicha supervisión, en caso de ser realizada por una persona, podría perder la atención fácilmente por cansancio o despiste, con los consiguientes riesgos que ello pueda ocasionar al trabajador y al proceso.

Un ejemplo de sistemas de control de lazo cerrado podría ser el de un regulador de nivel de agua en un depósito que posee una boya haciendo que, cuanto más cerca esté del nivel requerido para esa boya, el chorro de agua de llenado pierde presión. También un invernadero donde se controla la entrada de luz mediante un sensor es un sistema de lazo cerrado, incluso, es más, si el microondas fuera capaz de reconocer a la temperatura a la que esos alimentos están ya antes de calentarlos y sabiendo eso, pudiera mantener la temperatura o subirla, también sería un ejemplo de sistema de control de lazo cerrado.

Las características de los sistemas de control de lazo cerrado son:

- Ser complejos y amplios en cantidad de parámetros.

- La salida se compara con la entrada y le afecta para el control del sistema.

- Su propiedad de retroalimentación.

- Ser más estable a las posibles perturbaciones y variaciones internas.

El controlador es una componente del sistema de control que detecta los desvíos existentes entre el valor medido por un sensor y el valor deseado o también llamado “set point”, programado por un operador, emitiendo una señal de corrección hacia el actuador.

En el ejemplo de la válvula del regulador de nivel de agua comentado anteriormente, podríamos ver cada elemento del sistema de control a través del siguiente diagrama:

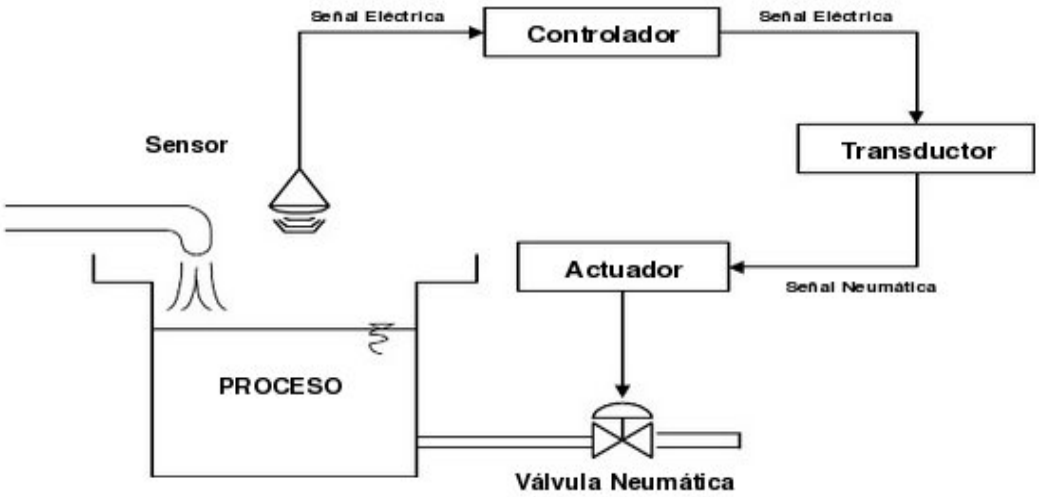


Ilustración 81: Representación gráfica ejemplo válvula de agua

De forma genérica:

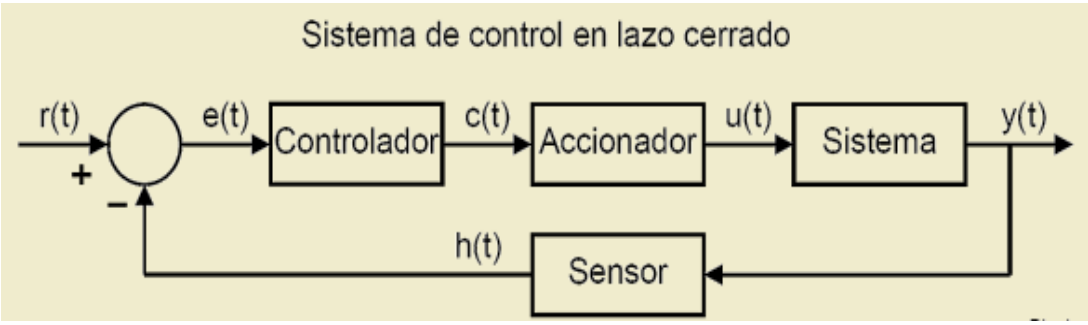


Ilustración 82: Esquema de sistema de control lazo cerrado

Dónde

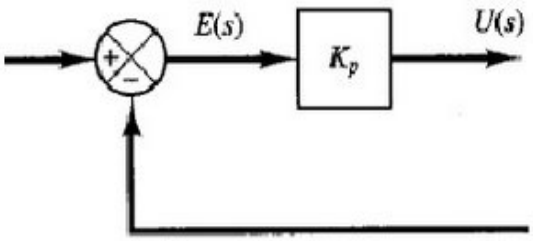
r(t) Referencia. Es el estado que se desea alcanzar en el sistema.
e(t) Error. Es la diferencia entre el estado deseado y el estado real del sistema a controlar.
c(t) Control. Es la señal que genera el controlador.
u(t) Accionamiento.
y(t) Salida del Sistema. Es el estado real que ha alcanzado el sistema a controlar.
h(t) Realimentación. Es la medida del estado del sistema.

Podríamos definir un actuador como un dispositivo mecánico cuya función es proporcionar fuerza para mover o "actuar" otro dispositivo mecánico. En un sistema de control, según sea la forma en que conteste el actuador, distinguiremos distintos tipos de acciones de control, algunas de esas acciones serán llamadas básicas aunque lo más común es que respondan mediante una combinación de dichas acciones básicas.

ACCIONES BÁSICAS	COMBINACIÓN DE ACCIONES BÁSICAS
Proporcional (P)	**Proporcional – Integrador (PI)**
Derivativa (D)	**Proporcional – Derivativa (PD)**
Integral (I)	**Proporcional – Integral - Derivativa(PID)**

Control Proporcional

En este tipo de control la magnitud de salida del controlador es proporcional a la magnitud del error (desviación entre el punto de medida y el valor consigna), es decir, si el elemento de control por ejemplo fuera una válvula, ésta recibiría una señal que es proporcional a la magnitud de la corrección requerida. Internamente la acción proporcional multiplica la señal de error por una constante Kp. En esencia, podría definirse este controlador como un amplificador con una ganancia ajustable.



$$\frac{U(s)}{E(s)} = K_p$$

Ilustración 83: Diagrama de bloques del controlador proporcional

Control Derivativo

En este sistema de control, la velocidad de cambio de la señal de entrada se utiliza para determinar el factor de amplificación, calculando la derivada de la señal. Es decir, el control derivativo puede dar una respuesta específica cuando existen cambios rápidos de la señal de error. La acción de control puede anticipar un error basado en la velocidad de respuesta. De este modo, mediante la derivada de la señal de error "conoce" sus características dinámicas (crecimiento o decrecimiento), produciendo una corrección antes de que la señal de error sea excesiva. Sin embargo, el control derivativo no puede utilizarse en solitario porque es incapaz de responder a una señal de error constante.

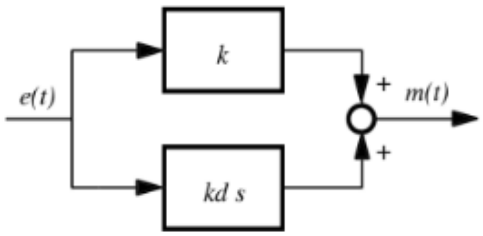


Ilustración 84: Diagrama de bloques del controlador derivativo

Control Integral

En este sistema de control, el valor de la acción de control es proporcional a la integral de la señal del error, por lo que, en este tipo de control, la acción varía en función de la desviación de la salida y del *tiempo* en que se mantiene esta desviación. Básicamente consiste en ir añadiendo pequeñas sumas (integral) para corregir el error y mantenernos cerca del nivel de consigna.

La velocidad de respuesta del sistema dependerá del valor de la pendiente de la rampa de acción integral. El inconveniente de este sistema de control es que la respuesta inicial es muy lenta y, el controlador no empieza a ser efectivo hasta haber transcurrido un cierto tiempo. En cambio, anula el error remanente que representa el controlador proporcional.

Matemáticamente podríamos entender el controlador con acción integral como:

$$\frac{du(t)}{dt} = K_i e(t)$$

$$u(t) = K_i \int_o^t e(t)$$

donde podemos ver que, el valor de la salida del controlador u(t) se cambia a una razón proporcional a la señal de error e(t) y dónde aparece una constante llamada Ki que será una constante ajustable.

$$U(s) = \frac{K_i}{s} E(s)$$

$$\frac{U(s)}{E(s)} = \frac{K_i}{s}$$

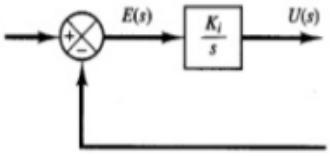


Ilustración 85: Diagrama de bloques del controlador integral

Control Proporcional – Integral

Es inusual controladores que actúen solamente con la acción integral, siempre actúan en combinación con reguladores de una acción proporcional, consiguiendo de esta forma, complementarse los dos tipos de reguladores, primero entra en acción el regulador proporcional de manera instantánea mientras que la acción integral actúa durante un intervalo de tiempo (Ti). Matemáticamente el control Proporcional Integral responde a la siguiente ecuación:

$$\frac{U(s)}{E(s)} = K_p\left(1 + \frac{1}{T_i s}\right)$$

donde Kp y Ti son parámetros que se pueden modificar según las necesidades del sistema. Si Ti es grande la pendiente de la rampa, correspondiente al efecto integral será pequeña y, su efecto será atenuado.

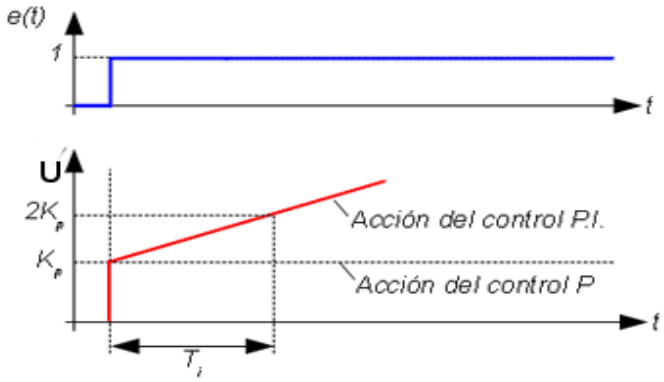


Ilustración 86: Grafica comportamiento controlador PI

Por lo tanto, la respuesta de un sistema de control que utilice una acción Proporcional – Integral, será la suma de las respuestas debidas a un control proporcional P, que será instantánea a detección de la señal de error, y con un cierto retardo entrará en acción el control integral I, que será el encargado de anular totalmente la señal de error.

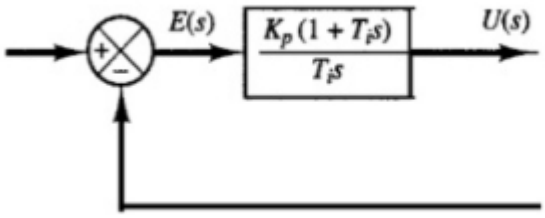


Ilustración 87: Diagrama de bloques del controlador PI

Para conocer más sobre otros Sistemas de Control de Lazo Cerrado ver la información adjunta en los anexos de esta memoria al final de la misma.

4.2.3.12. Movimiento del robot: Sistema de Control Elegido.

El proyecto inicialmente en su entrega estaba bajo un sistema de control de lazo abierto. Por esta misma razón, las perturbaciones y la necesidad de recalibrar el sistema podrían causar errores. El robot en aquel momento era sensible a perturbaciones externas y a variaciones internas de parámetros del sistema.

Dicho motivo es el que ha llevado a tomar la decisión de implementar en este proyecto un sistema de control de lazo cerrado que permitirá conocer el estado del sistema y corregir las desviaciones para que se pueda conseguir la respuesta deseada.

El primer paso fue escoger una acción de control proporcional porque, idealmente, para el control de posición de motores de corriente continua, cuya entrada es el voltaje aplicado sobre el inducido del motor y cuya salida es la posición de su eje de giro, basta con una acción de control proporcional para llegar al valor de referencia predefinido.

En la práctica esto no es posible debido a la presencia de una zona muerta de actuación de los motores. Esta no linealidad genera un error en el estacionario, el cual debe ser corregido mediante una acción integral.

Cuando el error es grande, la acción de control es grande y tiende a minimizar el error del sistema. Nos interesa que el robot se desplace lo más exacto a la distancia deseada y que el margen de error sea el menor posible.

Es en este momento cuando utilizamos la constante Kp llamada también como la constante de proporcionalidad, la cuál comentamos antes y que, de algún modo, determinará el grado de amplificación del elemento de control. Esta acción de control intenta minimizar el error del sistema. Cuando el error es grande, la acción de control es grande y tiende a minimizar este error. Cuando estamos ampliando la acción proporcional (Kp), se obtienen los siguientes efectos:

1) Aumenta la velocidad de respuesta.
2) Disminuye el error del sistema en régimen permanente.
3) Aumenta la inestabilidad del sistema.

Los dos primeros efectos son positivos pero el tercero no lo es y entonces nos encontramos con la situación que no podemos aumentar la Kp en los valores que nos venga en gana, cuanto más grandes mejor. Nos vemos forzados pues, a buscar un punto de equilibrio en el que al aumentar la Kp nos encontramos un punto en el que se consigue suficiente rapidez de respuesta del sistema y una reducción del error sin que el sistema sea demasiado inestable. Si por el contrario, reducimos la Kp nos encontraremos una reducción de velocidad

del sistema y un aumento del error permanente.

Las siguientes gráficas me ayudarán a explicar lo que sucede con Kp para que se entienda mejor y el motivo por el que se selecciona el valor que he utilizado en el proyecto.

Supongamos que queremos alcanzar el valor de referencia “150” en la siguiente gráfica:

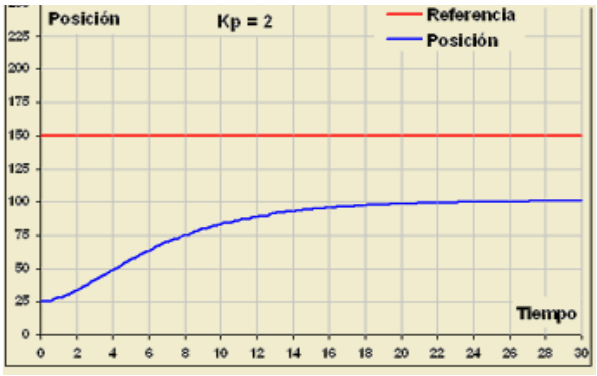


Ilustración 88: Gráfica comportamiento sistema control Proporcional

Si metemos una Kp = 2 y observamos la Posición, no alcanzaremos un valor de aproximación hacia la referencia ni habiendo transcurrido 24 unidades de tiempo. Con 28 unidades de tiempo aproximadamente, nos mantendremos a 50 unidades de distancia entre la referencia y la posición (150 - 100). Incrementamos Kp para ver si mejora.

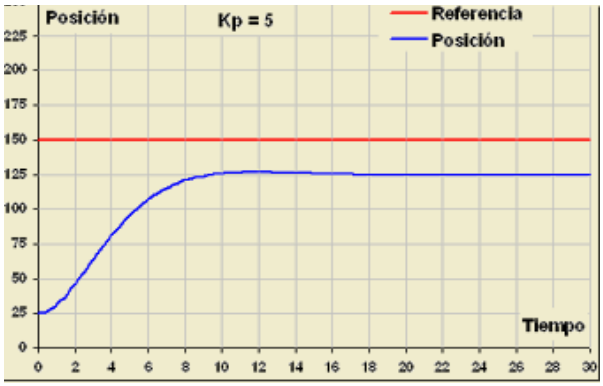


Ilustración 89: Gráfica comportamiento control Proporcional Kp=5

Con un valor de Kp = 5 hemos alcanzado un valor más cercano en menos tiempo. Aumentamos a ver si mejora.

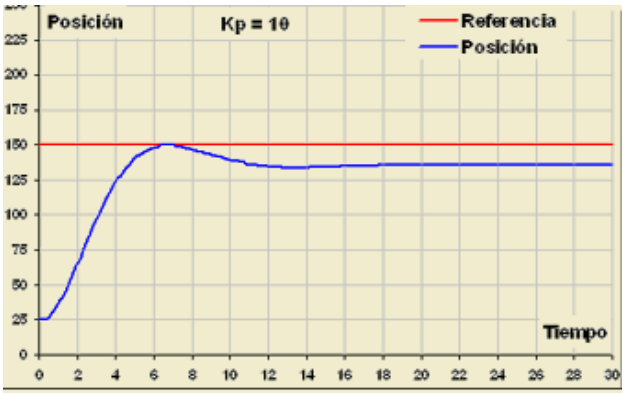


Ilustración 90: Gráfica comportamiento control Proporcional Kp=10

Con un valor de Kp = 10 hemos alcanzado el valor de referencia tardando menos tiempo y luego no quedarnos muy lejos del objetivo. Es posible que se obtenga un valor de Kp más “eficiente”. Aumentamos.

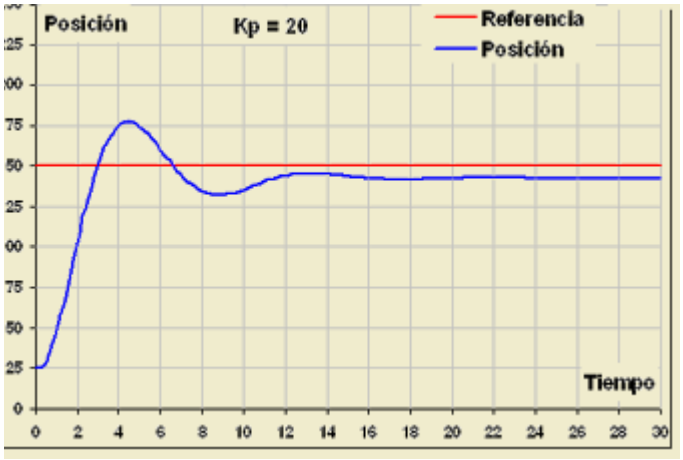


Ilustración 91: Gráfica comportamiento control Proporcional Kp=20

Con una Kp = 20 hemos alcanzado en menos tiempo el objetivo y luego se ha mantenido, a pesar de las oscilaciones que ya comienzan a observarse, bastante más cerca que los valores anteriores. Podríamos pensar que si aumentamos la Kp podríamos mejorar pero veamos que ocurre a valores más altos.

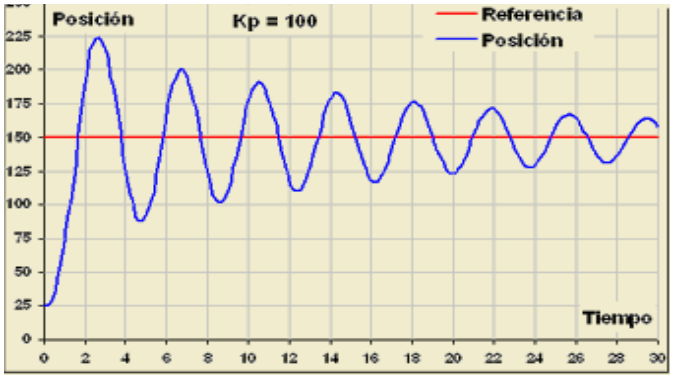


Ilustración 92: Gráfica comportamiento control Proporcional Kp = 100

Con una Kp = 100 lo que ocurre es que la posición se sobrepasa de la referencia e intenta estabilizarse, lo cuál provoca que existan más oscilaciones en el sistema. Es por tanto que, que debemos elegir una Kp aceptable para evitar el mayor número de oscilaciones posibles.

Con el valor de Kp = 20 estamos cerca de conseguir el objetivo pero aún falta alrededor de unas 5 – 10 unidades para alcanzar la medida que estamos

buscando. Para conseguir alcanzar el valor de referencia utilizaremos pequeñas sumas de pequeños errores. Sumar pequeñas porciones de medida implica el uso de la integral. Por tanto, precisamos de una acción de control integral.

Aumentando la acción integral Ki explicada en aparatados anteriores, obtendremos los siguientes efectos:

1) Disminuye el error del sistema en régimen permanente.

2) Aumenta las oscilaciones en el sistema.

3) Aumenta un poco la velocidad del sistema.

Como en el control proporcional, no podemos dar valores muy altos a la constante Ki porque haríamos que el sistema no obtuviese los resultados esperados. Gráficamente esto se traduce como:

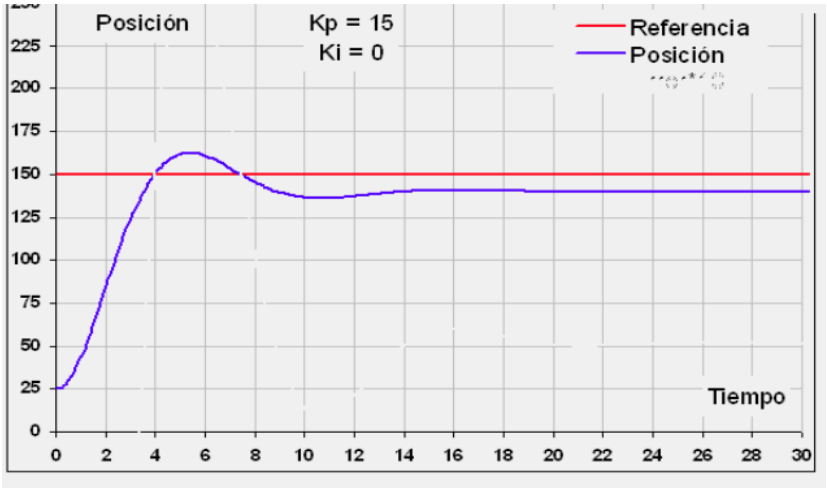


Ilustración 93: Gráfica comportamiento control PI Kp=15 y Ki = 0

Supongamos que nos hemos quedado con un valor Kp = 15 y no 20 como en las anteriores gráficas. Entonces, si Ki = 0, la acción integral no se aplica y por tanto el sistema se comporta como un sistema de control proporcional.

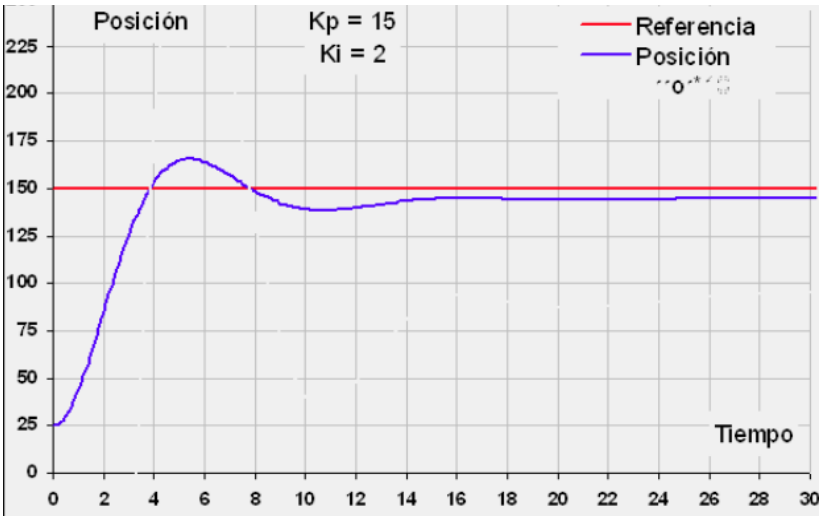


Ilustración 94: Gráfica comportamiento control PI Kp=15 y Ki = 2

Con un valor de Ki=2 nos aproximamos más al valor de referencia reduciendo con ello el error que finalmente se hará cero en algún momento durante el transcurso del tiempo.

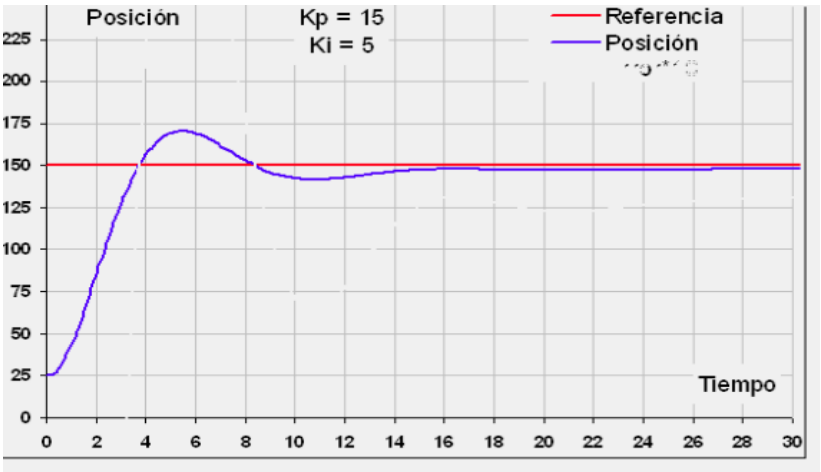


Ilustración 95: Gráfica comportamiento control PI Kp=15 y Ki = 5

Con un valor de Ki = 5 mejoramos la respuesta, sin embargo, si aumentamos esta cantidad hasta Ki = 10 obtenemos:

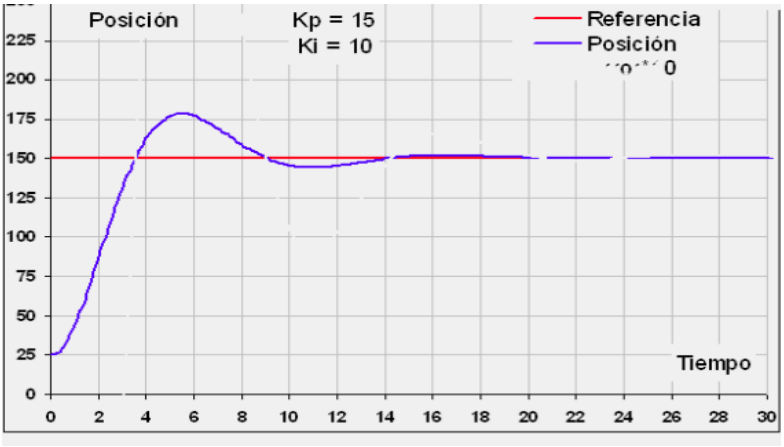


Ilustración 96: Gráfica comportamiento control PI Kp=15 y Ki = 10

Podríamos pensar que si aumentamos más la constante, obtendremos mejores resultados, sin embargo, esto no ocurre:

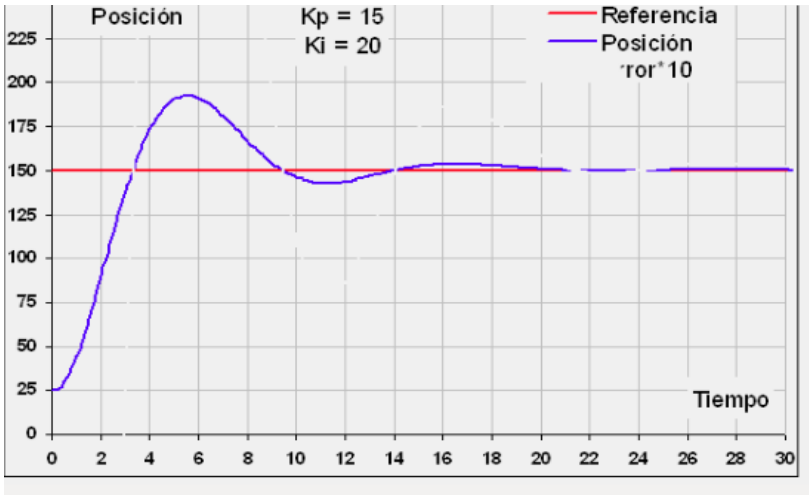


Ilustración 97: Gráfica comportamiento control PI Kp=15 y Ki = 20

Podemos ver que, para valores de Ki = 20 ya comienza haber oscilaciones en el sistema, si ampliamos ese valor hasta 50, las oscilaciones son mucho mayor.

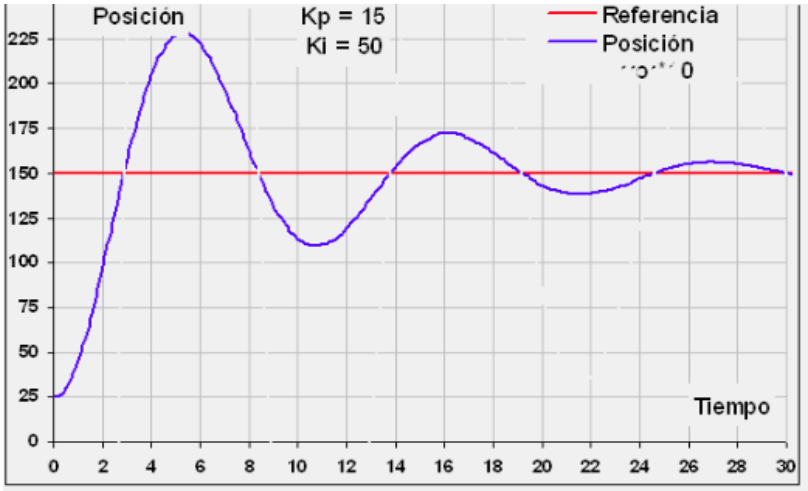


Ilustración 98: Gráfica comportamiento control PI Kp=15 y Ki = 50

No utilizamos un Sistema de Control Derivativo por dos motivos principalmente, porque nos basta con un sistema de control proporcional – integral y porque el comportamiento actual del robot tras usar el sistema de control PI, no da muestras de grandes oscilaciones, síntomas de un funcionamiento inesperado en un sistema de control.

Gráficamente en un ejemplo, el comportamiento de un sistema de control Derivativo sería el siguiente:

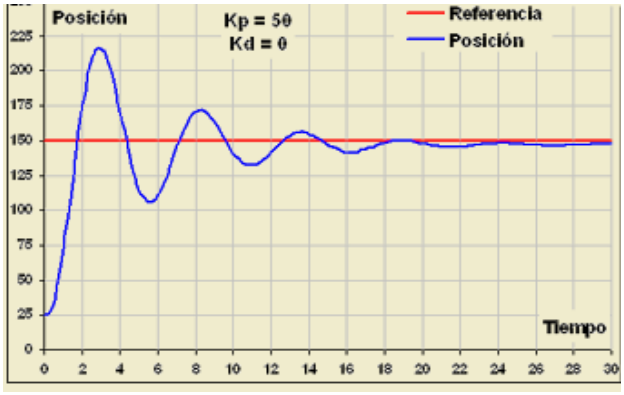


Ilustración 99: Gráfica comportamiento control PD Kp=50 y Kd = 0

Partimos de un valor Kp alto para poder ver la mejora del sistema frente a las oscilaciones.

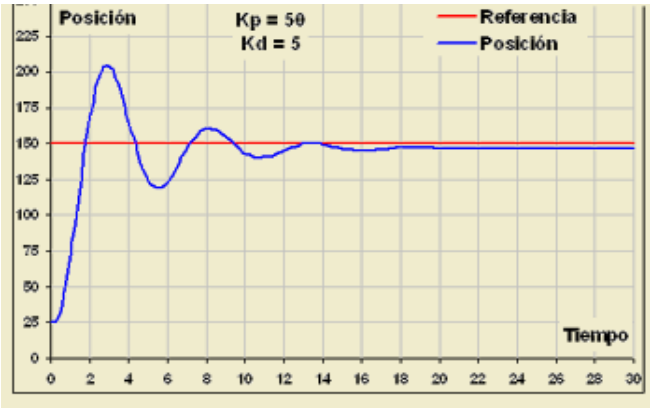


Ilustración 100: Gráfica comportamiento control PD Kp=50 y Kd = 5

Según incrementamos Kd vamos viendo como las oscilaciones decrecen. Con un valor de Kd de 10 que apenas dista del anterior valor, observamos la mejora. Las Oscilaciones en la respuesta del sistema comienzan a suavizarse lentamente.

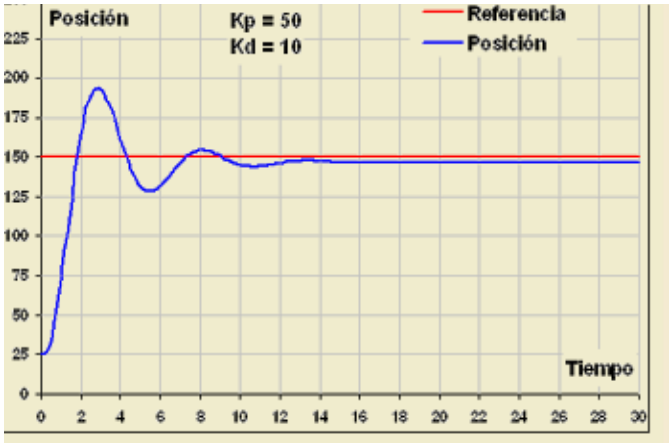


Ilustración 101: Gráfica comportamiento control PD Kp=50 y Kd = 10

Para valores de Kd = 20 el sistema ya ha mejorado muchísimo en una respuesta en valores de tiempo bajos. Obsérvese que las oscilaciones se van suavizando.

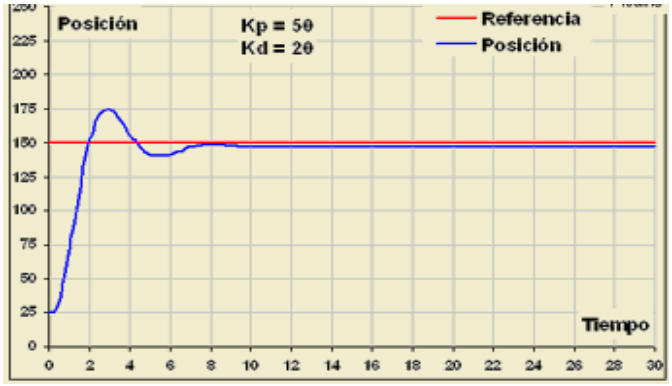


Ilustración 102: Gráfica comportamiento control PD Kp=50 y Kd = 20

En valores altos de Kd, en este caso, con un valor igual que la constante de proporcionalidad tenemos el siguiente comportamiento:

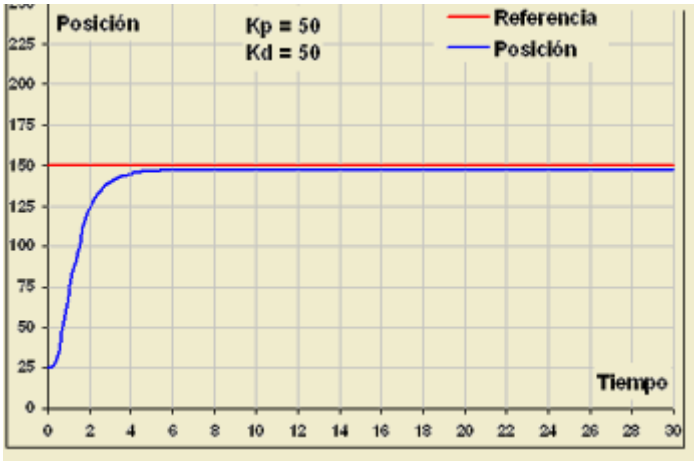


Ilustración 103: Gráfica comportamiento control PD Kp=50 y Kd = 50

La acción derivativa ha conseguido, en poco tiempo, estabilizarse muy cerca del valor de referencia que estamos buscando pero como podemos ver, nunca lo alcanza. Para poder alcanzar ese valor de referencia es necesario la acción integral. Por otro lado, puede observarse que han desaparecido aquellas oscilaciones que podrían mostrar un comportamiento inesperado dentro del sistema de control.

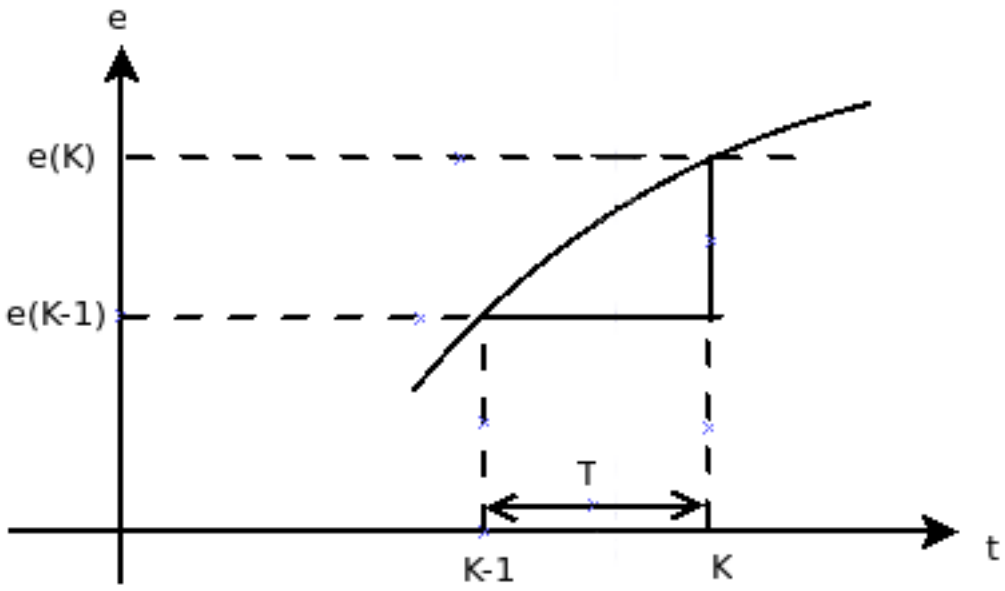
El comportamiento del robot a su llegada al objetivo debe ser lo más estable posible por lo que, observándolo durante su trayectoria, se puede apreciar una reducción de velocidad hasta quedar completamente parado. Es en ese momento cuando podemos observar que está actuando la acción integral, provocando ese "pequeño empuje" que le hace llegar al objetivo.

Por tanto, el sistema de control que está utilizando el robot móvil es un sistema de control proporcional – integral. Matemáticamente tenemos lo siguiente:

$$u(t) = Kp\, e(t) + Ki \int_0^t e(t')\, dt'$$

Debido a que queremos implementar esta acción de control en un dispositivo digital, debemos obtener una expresión discreta de dicha acción. Para ello, debemos hacer que las aproximaciones necesarias para permitir aproximar las acciones integral y derivativa del controlador.

Observando el comportamiento en la siguiente gráfica:



Deducimos la siguiente expresión:

$$u(k)=Kpe(k)+Ki\sum_{n=0}^{K}Te(n-1)+T\frac{(e(k)-e(k-1))}{2}$$

Simplificando y dejando T/2 fuera del sumatorio, obtenemos

$$Ki\frac{T}{2}\sum_{n=0}^{K-1}(e(n)+e(n-1))$$

por lo que hemos conseguido que

$$U(k-1)=Kpe(k-1)+Ki\frac{T}{2}\sum_{n=0}^{K-1}(e(n)+e(n-1))$$

una vez obtenida u(k-1) procedemos a restársela de u(k) para eliminar el sumatorio

$$u(k)-u(k-1)=Kp\,e(k)-Kp\,e(k-1)+Ki\left(\frac{T}{2}\right)(e(k)+e(k-1))$$

luego

$$u(k)=u(k-1)+\left(Kp+\frac{KiT}{2}\right)e(k)+\left(-Kp+\frac{KiT}{2}\right)e(K-1)$$

esta fórmula será la que utilizaremos para nuestro sistema de control

y llamaremos estos dos términos como p0 y p1

$$\left(Kp+Ki\frac{T}{2}\right) = p0$$

$$\left(-Kp+Ki\frac{T}{2}\right) = p1$$

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

SECCIÓN DE INGENIERÍA INFORMÁTICA

Capítulo 5

Software: Estrategia y Pseudocódigo

Titulación: Ingeniería en Informática

Alumno: Francisco Raúl Machín Castilla

Tutor: Santiago Torres Álvarez

Julio 2016

Contenido

5 SOFTWARE

5.1. Introducción

En este apartado del proyecto se explicará de manera breve la funcionalidad del prototipo reflejado en el código de Python.

El programa se encuentra dividido en funciones las cuáles, recibirán o no, ciertos parámetros y retornarán valores que precisamos para su ejecución.

A continuación y de manera resumida, se detalla la parte más importante del código en Python que se ejecuta en el prototipo.

5.2. Estrategia y Pseudocódigo

La estrategia actual para el control del movimiento del robot que se ha seguido está basada principalmente en el uso de dos hilos en Python, uno de ellos estará pendiente de las cuentas de los contadores para saber cuántos ticks hace falta desplazar para alcanzar el objetivo e ir contándolos durante el recorrido y otro hilo que se encargará de todo lo relacionado con el sistema de control, generar el control, aplicar condición de control, chequear la condición de parada, etc.

Observemos que ambas tareas (conteo y control) deben realizarse al mismo tiempo, de ahí el uso de la concurrencia.

5.2.1 Hilo o Thread en Python

Un hilo de ejecución es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo.

La creación de un hilo permite a una aplicación poder realizar de manera concurrente, es decir, varias tareas simultáneamente. Podríamos definir también a un hilo como una tarea que puede ser ejecutada al mismo tiempo con otra tarea. Los distintos hilos comparten una serie de recursos tales como el espacio de memoria, archivos abiertos, situación de autenticación, etc.

Existen dos formas de utilizar la concurrencia en Python: threading and multiprocessing. La diferencia entre la primera y la segunda es que, en threading los hilos se ejecutan en el mismo espacio de memoria mientras que en la segunda, los procesos tienen memoria separada.

En este proyecto se utilizará la primera (threading) y para ello tendremos que hacer uso del módulo threading para poder utilizar la clase Thread.

```
import threading, time, sys, signal, math, os, RPIO
import RPi.GPIO as GPIO
from RPIO import PWM
```

Ilustración 104: código Python para la importación de las librerías

Básicamente lo único que tenemos que hacer es pasarle al hilo como parámetros, la función que queremos ejecutar y los parámetros con los que queremos ejecutarla. A continuación, basta con usar el método start del hilo y usar los métodos que necesitemos para llevar a cabo lo que pretendemos en nuestro programa.

```
852 def _handle_signal(signal, frame):
853         global running
854         running = False
855
856 if __name__ == '__main__':
857         signal.signal(signal.SIGTERM, _handle_signal)
858         signal.signal(signal.SIGINT, _handle_signal)
859
860 hilo = threading.Thread(target=monitor, args=(1,), kwargs={'itemID':'1', 'threshold':60})
861 hilo2 = threading.Thread(target=actualizo_contador, args=(1,), kwargs={'itemID':'2', 'threshold':60})
862 hilo2.start()
863 hilo.start()
864 hilo.join(30)
865 hilo2.join(30)
866
```

Ilustración 105: código Python para el uso de threads

Cómo puede observarse los dos hilos en el código de este proyecto los hemos definido en las siguientes líneas de código:

```
hilo = threading.Thread(target=monitor, args=(1,), kwargs={'itemID':'1', 'threshold':60})
```

Hemos creado un hilo utilizando el módulo “threading” extendiendo una clase Thread para poder crear nuestros propios hilos de ejecución que son hilo e hilo2.

Cada hilo tiene el código que queremos que ejecute el thread. Es decir, el hilo ejecutará el código que hay en la función “monitor” y el hilo2 ejecutará el código que hay en la función “actualizo_contador”. Con “args” mapeamos todos los

argumentos posicionales a una tupla llamada args. Y con kwargs mapeamos todos los argumentos de palabra clave a un diccionario llamado kwargs.

Ejecutamos el comienzo de cada hilo con el método start() y después el método join() para que finalicen los hilos.

Es posible que quisiéramos para un hilo enviando una señal de control a través del teclado, en este caso utilizando la combinación Crtl + C para cortar el funcionamiento del hilo. Por esto utilizamos

```
signal.signal(signal.SIGINT, __handle_signal)

signal.signal(signal.SIGTERM, __handle_signal)
```

es una señal que se envía al proceso para comunicarle un apagado cerrando bien las conexiones, ficheros y limpiando sus propios buffers, es decir, no haciendo un apagado forzoso.

5.2.2 Sistema de Control Proporcional Integral

Una de las partes más importantes de este proyecto es el **sistema de control** que estamos utilizando. Debemos aplicar el sistema de control hacia ambos motores del robot así que se tuvo que implementar el código para ambas ruedas, recordemos que estamos implementando la siguiente fórmula para el control y que se encargará de ello el primer hilo que hemos declarado, hilo “monitor”:

$$u(k)=u(k-1)+\left(Kp+\frac{KiT}{2}\right)e(k)+\left(-Kp+\frac{KiT}{2}\right)e(K-1)$$

dónde hemos dicho que estos términos son p0 y p1

$$\left(Kp+Ki\frac{T}{2}\right) = p0$$

$$\left(-Kp+Ki\frac{T}{2}\right) = p1$$

Por tanto, la implementación del código se realizó como se indica en la siguiente imagen:

```
cont_aux = 0

tole_error = 2
T = 0.05

ukm1_der = 0.0
ukm1_izq = 0.0
ekm1_d = 0.0
ekm1_i = 0.0

#VALORES A KsuP y KsuI para conseguir movimiento
#kp = 0.10
kp = 0.75
ki = 0.01 * kp


p0 = (ki * T/2) + kp

#p0 = kp
#p1 = ki * T - kp

p1 = (ki * T/2) - kp
```

Ilustración 106: código Python declaración de valores iniciales para sistema control

Primero la declaración e iniciación de variables. Definición de un margen de error, digamos una tolerancia al error. Definición de un contador auxiliar y de un tiempo T que explicaremos más adelante. Ajustes de las constantes de entrada Kp y Ki para el sistema de control y definición de los dos término explicados anteriormente p0 y p1.

A continuación, el cálculo del error para cada rueda que será el valor de referencia menos la posición seguido del cálculo de la fórmula:

```
ek_d = num_ticks_der - abs(contador2)

uk_d = ukm1_der + p0 * ek_d + p1 * ekm1_d
```

```
731         else:
732                 #print "GIRO_DERECHA"
733                 Motor1 = 28
734                 Motor2 = 31
735
736         GPIO.setup(Motor1, GPIO.OUT)
737         p_izq = GPIO.PWM(Motor1, 100)
738         p_izq.start(0)
739
740         GPIO.setup(Motor2, GPIO.OUT)
741         p_der = GPIO.PWM(Motor2, 100)
742         p_der.start(0)
743
744         freq_muestreo = T * 10000
745         while running:
746                 cont_aux = cont_aux + 1
747                 if (math.fmod(cont_aux, int(freq_muestreo))  == 0):
748
749                         ek_d = num_ticks_der - abs(contador2)
750                         uk_d = ukm1_der + p0 * ek_d + p1 * ekm1_d
751                         ek_i = num_ticks_izq - abs(contador1)
752                         uk_i = ukm1_izq + p0 * ek_i + p1 * ekm1_i
753
754                         uk_i_s = min(uk_i,100.0)
755                         uk_i_s = max(uk_i_s, 0.0)
756                         uk_d_s = min(uk_d,100.0)
757                         uk_d_s = max(uk_d_s, 0.0)
758
759                         uk_max = max(uk_i_s, uk_d_s)
760                         p_izq.ChangeDutyCycle(int(uk_max))
761                         p_der.ChangeDutyCycle(int(uk_max))
762                         print "ek_i = %s, uk_i = %s, ek_d= %s, uk_d= %s" % (ek_i, uk_i, ek_d, uk_d)
763                         print "-------------------------------------------------------------------------------------"
764                         print "ek_d = %s, uk_d = %s, ekm1_d=%s, ukm1_der = %s" % (ek_d, uk_d, ekm1_d, ukm1_der)
765                         print "====================================================================================="
766
767                         #PARA EL GIRO la condicion de parada distinta
768                         if (sentido == "IZQUIERDA") or (sentido == "DERECHA"):
769                                 if (max(ek_i, ek_d) < tole_error):
770                                         p_der.stop()
771                                         p_izq.stop()
772                                         GPIO.cleanup()
773                                         break
774
775
)-CODIGO
778                         elif (sentido == "ADELANTE") or (sentido == "ATRAS"):
779                                 if (abs(ek_i) < tole_error) or (abs(ek_d) < tole_error):
780                                         p_der.stop()
781                                         p_izq.stop()
782                                         GPIO.cleanup()
783                                         break
784
                        #actualizamos variables
                        ukm1_der = uk_d
                        ukm1_izq = uk_i
```

Ilustración 107: código Python que implementa el sistema de control

En el código se observa que existe una comprobación dentro del bucle para poder poner en marcha el sistema de control:

```
if (math.fmod(cont_aux, int(freq_muestreo)) == 0):
```

Previo a esta condición se declara la variable “cont_aux” y una freq_muestreo = T * 10000. Esto se debe a que la Frecuencia con la que trabaja la Raspberry Pi es demasiado rápida. Necesitamos aplicar el control a una frecuencia menor que a la que la placa utiliza. **Debemos aplicar el control en intervalos regulares de tiempo** y por este motivo se ha creado la variable T con el valor 0.05. La variable cont_aux se incrementa y solo se aplica la acción de control cuando la frecuencia de muestreo y el contador son múltiplos entre sí.

Ahora escalamos entre 0 y 100 los valores para saturar la entrada cuando llamemos al ChangeDutyCycle que modificará la velocidad de giro de los motores aplicando la PWM. La llamada al cambio de ciclo de trabajo se hace en porcentajes (0 – 100%) como ya expliqué en apartados anteriores de esta memoria.

Las condiciones de parada para el giro es diferente que para el recorrido de la distancia. En el primero elegiremos el mayor de los errores para comparar con el valor de tolerancia del error para detener la ejecución del bucle while para esperar que las dos ruedas lleguen al objetivo y en la segunda, en el desplazamiento, nos basta con que uno de ellos llegue para detener la ejecución.

Finalmente actualizamos las variables para que sigamos iterando en el resto de valores y detenemos la ejecución del hilo que se encarga de actualizar los contadores, el hilo2 llamado “actualizo_contador”:

```
787
788                         ekm1_d = ek_d
789                         ekm1_i = ek_i
790
791         #print "ek_i = %s, uk_i = %s, ek_d= %s, uk_d= %s" % (ek_i, uk_i, ek_d, uk_d)
792         hilo2.stopped =True
793         running = False
794         #end_WHILE
795
```

Ilustración 108: código Python que implementa sistema de control (2)

Precisamos que la Raspberry Pi esté en todo momento contando los ticks que van haciendo que el robot se desplace, tanto para ir hacia adelante o atrás o para realizar un giro. De este cometido se encarga el hilo2 “actualizo_contador” el cuál en código tiene la siguiente apariencia:

```
#SEGUNDO HILO para actualizar los contadores llamando al movimiento detras, delante o giro
def actualizo_contador(tid, itemID=None, threshold=None):
        global sentido
        global contador1
        contador1 = 0
        global contador2
        contador2 = 0

        global PI
        PI = 3.14

        global dos_pi_R
        dos_pi_R = 34.5575

        global TOTAL_TICKS_VUELTA_IZQ
        TOTAL_TICKS_VUELTA_IZQ = 86
        global TOTAL_TICKS_VUELTA_DER
        TOTAL_TICKS_VUELTA_DER = 89

        global tolerancia
        tolerancia = abs(TOTAL_TICKS_VUELTA_DER - TOTAL_TICKS_VUELTA_IZQ)

        global TICKS_POR_GRADO_IZQ
        TICKS_POR_GRADO_IZQ = 0.3752 * 2

        global TICKS_POR_GRADO_DER
        TICKS_POR_GRADO_DER = 0.3883 * 2

        if (sys.argv[1] == 'F'):
                sentido = "ADELANTE"
                resultados = adelante()

        elif (sys.argv[1] == 'B'):
                sentido = "ATRAS"
                resultados = atras()
                #servo = 'D'
                #accion_servo(servo)

        elif (sys.argv[1] == 'L' or sys.argv[1] == 'R'):
                sentido = "IZQUIERDA"
                if (sys.argv[1] == 'R'):
                        sentido = "DERECHA"
                resultados = giro()

        RPIO.cleanup()
        print (resultados)
```

Ilustración 109: código Python del hilo "actualizo_contador"

Actualizo_contador se encarga de declarar e inicializar las variables de los contadores, declarar los valores de las variables globales PI, dos_pi_R que recordemos, era la distancia que se desplazaba el dar una vuelta completa cada rueda en centímetros, el total de ticks en dar esa vuelta completa cada rueda, la tolerancia que es el valor que hay de diferencia entre los ticks de una rueda y la otra, existen 3 ticks de tolerancia y el número de ticks por cada grado que gira el robot y que explicaremos a continuación de dónde se obtuvo.

Finalmente, actualizo_contador dependiendo de la lectura del movimiento que hayamos pedido al robot que realice, ejecuta un desplazamiento (adelante o atrás) o un giro (izquierda o derecha).

Cada vez que pidamos al robot una acción de desplazamiento, entraremos en el código en las funciones de adelante, atrás o giro, explicadas en apartados anteriores, dónde realizará una lectura de los encoders devolviendo

el número de ticks contados y parando el primer hilo declarado “monitor”.

En el código puede observarse que la ejecución de un hilo al terminar, para la ejecución del otro y viceversa. Esto se debe a la lógica seguida en la estrategia utilizada.

5.2.3 Servos

En el apartado de alimentación para los servos (4.2.3.8) de este proyecto se explican qué son los servomotores que el prototipo posee para la recogida y almacenaje de las pelotas de tenis.

Recordemos que los servomotores son dispositivos similares a un motor DC que tienen la capacidad de ubicarse en cualquier posición dentro de su rango de operacioń y mantenerse estable en esa posición. Los servomotores se controlan por medio de un pulso de ancho variable y frecuencia constante (PWM). La duración de ese pulso determina el ángulo girado. Su posición neutral suele estar en torno a los 1,5 ms.

El software utilizado para el mecanismo de los servos no conlleva mucha complicación. Gracias la librería RPIO podemos importar el módulo PWM con el que utilizaremos justo lo que necesitamos para los Servomotores.

```
19 # Funcion que activa los servos correspondientes para recoger y soltar las pelotas de tenis
20 def accion_servo(servo_mover):
21
22         servo = PWM.Servo()
23         #activamos servo delantero para recoger pelotas (sube 90 - 40 grados evitar choque varilla delantera suelo)
24         if (servo_mover == 'D'):
25                 servo.set_servo(27, 2500)
26                 time.sleep(2)
27                 servo.set_servo(27, 1100)
28                 time.sleep(1)
29                 servo.stop_servo(27)
30                 RPIO.setup(27, RPIO.IN)
31         #activamos servo trasero para soltar las pelotas ( 54 - 4 grados aprox)
32         else:
33                 servo.set_servo(23, 1500)
34                 time.sleep(1)
35                 servo.set_servo(23, 100)
36                 time.sleep(1)
37                 servo.stop_servo(23)
38                 RPIO.setup(23, RPIO.IN)
```

Ilustración 110: código Python para movimiento de los servos

Declaramos el objeto servo y recogemos en la variable “servo_mover” de que servo se trata, “D” para el servo delantero y si no, el servo trasero. Hemos elegido el pin en modo BCM 27 para el servo delantero en la GPIO y el pin 23 para el servo trasero.

Activamos el movimiento del servo haciendo llamada a la función “set_servo” a la cuál le pasamos el pin que queremos que se active y un valor en µs que determinará el ángulo que queremos que gire.

Por circunstancias del diseño del prototipo los valores de subida (recogida de pelotas) y bajada del gancho delantero, sin tocar el suelo, están en torno a unos 2500µs para la subida y 1100µs para la bajada. El gancho trasero que permite el vaciado de las pelotas de tenis desde el recinto habilitado para almacenarlas, está en torno a unos 1500µs para dejar caer las pelotas y 100µs para volver a cerrar el habitáculo.

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

SECCIÓN DE INGENIERÍA INFORMÁTICA

Capítulo 6

Resultados del proyecto y conclusiones

Titulación: Ingeniería en Informática

Alumno: Francisco Raúl Machín Castilla

Tutor: Santiago Torres Álvarez

Julio 2016

Contenido

6 RESULTADOS DEL PROYECTO Y CONCLUSIONES

6.1. Mejoras e innovaciones en lo referente al control

Entre las mejoras más importantes a destacar tenemos:

- Se ha actualizado la electrónica del robot adecuando el espacio disponible para los nuevos elementos.

- Se ha hecho omisión del uso del Arduino UNO que era el objetivo que perseguía el proyecto. El resultado de la mejora repercute en el espacio ocupado y también en la comunicación con el prototipo.

- Se han hecho reparaciones en el del motor derecho ya que presentaba deficiencias en su giro.

- Se ha eliminado el uso de una placa protoboard y de la circuitería de alimentación, eliminando también la cantidad de cableado y circuitos integrados, dejando de una forma más clara y visible.

6.2. Mejoras en la electrónica

Alguna de las mejoras más destacadas realizadas en la electrónica del robot son las siguientes:

- Se ha mejorado el circuito de control de los motores DC:

  - Se ha sustituido la circuitería de los puentes en H con **dos controladores** de motores doble puente H **en un circuito integrado L298N** y la circuitería donde estaban presente los diodos de rectificación 1N5819 que se utilizaban para tener una mejor respuesta ante los picos de tensión producidos por los motores DC. Con lo cuál hemos liberado espacio y peso. Las dimensiones reducidas de los circuitos integrados facilitan el manejo y conexión del resto de componentes.

  - Se ha eliminado la placa de alimentación que el prototipo traía consigo inicialmente. Dicha placa presentaba problemas de corriente, golpes y ralladuras en el circuito impreso. El **regulador de tensión** que la ha sustituido fue utilizado como recurso en poco tiempo pero ha contribuido enormemente dando estabilidad a la alimentación que necesitaba el robot y también liberando de espacio para añadir otros componentes. Inicialmente, el uso del regulador de

tensión se ideó para alimentar al mismo tiempo los servo-motores encargados de la recogida de las pelotas en la pista.

- Tras el consejo recibido por un profesor del departamento de electrónica, se añade el **Módulo LM2596** convertidor de Voltaje DC-DC Buck 1.25V-35V. Dicho módulo se añade por motivos de sobrecalentamiento del regulador de tensión que no viene preparado para solventar esa situación de exceso de calor en placa. El reducido tamaño y el uso limitado de cables, sólo 4 cables, hacen que sea un componente muy ligero, sencillo de usar y manejable.

6.3. Mejoras en la implementación de algoritmos y software

Las mejoras de software han sido las más importantes a destacar en la consecución de este proyecto por diversos motivos:

- Se precisaba de eliminar la comunicación serial que mantenían los dispositivos Raspberry Pi y Arduino UNO para obtener una mayor rapidez de respuesta en los movimientos del prototipo y un menor gasto de recursos.

- Tras la eliminación del Arduino UNO la estrategia en los algoritmos e implementación de los mismos cambiaba casi en su totalidad.

- El control de motores, lectura de encoders y resto de componentes se han traslado hacia la Raspberry Pi para que sea ella la encargada de manejar todo el prototipo ya que era el único dispositivo capaz de establecer el control.

- El control del prototipo estaba implementado en un sistema de control de lazo abierto que se tuvo que cambiar a un sistema de control en lazo cerrado para regular el correcto funcionamiento del prototipo reduciendo las probabilidades de fallo y obtener los resultados deseados.

Las mejoras software han sido:

- Uso del lenguaje Python por motivos de claridad de código debido a su sintaxis sencilla y sobretodo, porque es el lenguaje recomendado para Raspberry Pi. También por ser un lenguaje interpretado que te permite probar código antes de incorporarlo al script de tu programa. Por ser multiplataforma, dinámico, fuertemente tipado y porque es orientado a objetos. Y un motivo de peso importante es la cantidad de librerías que ya tiene implementadas facilitándonos tener que estar elaborándolas nosotros.

- El uso del lenguaje Python ha reducido en más de un 80% el código inicial escrito en Lenguaje C con el que inicialmente venía el proyecto para el control del movimiento del robot móvil.

- El conteo de los ticks del robot se realizaron manejando un diagrama de estados que en todo momento sirvió de orientación al prototipo. Es decir, si el robot, en la lectura de los encoders en un movimiento en concreto, se encontraba en un estado determinado y venía otro estado, sabía que para llegar a este, el conteo de los ticks estaba limitado a una cantidad conocida. Esto ayudaba al robot a saber en qué estado se encontraba.

- El uso de las librerías RPi.GPIO y RPIO facilitan de manera sencilla el uso de la comunicación a través de los pines GPIO de Raspberry Pi con cualquier otro mecanismo. A parte, las herramientas que aportan estas librerías, ambas sólo para Python, ayudan muchísimo en las tareas de testeo que estemos realizando con la placa.

- La utilización de hilos en Python permite realizar las tareas de manera concurrente sin necesidad del uso del Arduino UNO que inicialmente, liberaba del trabajo del movimiento de los motores y lectura de los encoders al prototipo.

- El uso de las librerías de Python también facilita poder manejar los servos para la recogida de pelotas y almacenaje de las mismas para luego depositarlas en el lugar destinado para ello. Ya no hace falta, por tanto, el uso de trabajar con señales PWM de frecuencias distintas para el manejo de los servos. Uno de los motivos por los que inicialmente el proyecto utilizaba Arduino UNO era éste.

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

SECCIÓN DE INGENIERÍA INFORMÁTICA

Capítulo 7

Posibles mejoras del proyecto

Titulación: Ingeniería en Informática

Alumno: Francisco Raúl Machín Castilla

Tutor: Santiago Torres Álvarez

Julio 2016

Contenido

7 POSIBLES MEJORAS DEL PROYECTO

7.1. Posibles Mejoras en el futuro.

De cara al futuro podrían tenerse en cuenta las siguientes mejoras:

- Mejorar el sistema de tracción: ruedas, reductoras y sobretodo motores utilizando un único motor que aporte movimiento para ambas ruedas ya que esto, corregiría los errores introducidos por las diferencias entre cada motor a la hora de girar individualmente o bien, introducir dos motores de mayor potencia lo más equilibrados posible.

- Mejorar los encoders utilizados con unos encoders integrados evitando los errores que generan los actuales por vibraciones o posibles choques con elementos externos durante la recogida de pelotas.

- Utilizar un sensor para la recogida de pelotas. En el momento de la recogida de pelotas utilizar un sensor que nos indicas que, efectivamente la recogida en el servo delantero ha sido un éxito y no hemos realizado una recogida en falso.

- Utilizar un PC OnBoard con mayor procesamiento. Durante la detección de pelotas, lectura de encoders y movimiento de motores se utilizará una capacidad mayor a la que la Raspberry Pi implantada en este proyecto puede realizar con eficiencia. Sustituir la placa por una un modelo más avanzado ayudaría para conseguir este objetivo. El modelo más actual en estos momentos es la Raspberry Pi 3 que ya viene con “quad-core” (cuatro núcleos) y 1.2GHz que, sin duda, ayudará a realizar más tareas al mismo tiempo (visión, lectura, movimiento).

- Utilizar un modelo de raspberry que soporte 802.11n Wireless LAN (wifi incorporada) nos eliminará el dispositivo USB que en este proyecto utiliza el prototipo para comunicarse con el ordenador central.

- Añadir una única fuente de alimentación capaz de suministrar el voltaje necesario para los componentes del circuito del robot: motores, circuitería en la protoBoard y servos.

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

SECCIÓN DE INGENIERÍA INFORMÁTICA

Capítulo 8

Referencias

Titulación: Ingeniería en Informática

Alumno: Francisco Raúl Machín Castilla

Tutor: Santiago Torres Álvarez

Julio 2016

Contenido

8 REFERENCIAS

8.1. Proyectos.

- Implementación software, mejoras electrónicas y mecánicas, y centralización de algoritmos y el reconocimiento de imágenes en el robot BallaBot-ULL. Autores: Adison Daniel Pérez Carballo y David Rodríguez Martín.

- Robot móvil autónomo para la recogida de pelotas de tenis: implementación mecánica y electrónica. Autor: Cesar Vera de Armas.

8.2. Referencias Web.

Algunas de las Referencias encontradas en Internet son:

http://www.realvnc.com/

http://www.python.org

https://es.wikipedia.org/wiki/Raspberry_Pi

https://es.wikipedia.org/wiki/Sistema_de_control

https://pypi.python.org/pypi/RPi.GPIO

https://github.com/metachris/RPIO

https://pythonhosted.org/RPIO/

http://www.bristolwatch.com/L298N/

http://blog-j.marcano.net.ve/index.php/page/8/?nggpage=3&pageid=255

https://github.com/nebrius/raspi-io/issues/45

http://www.toptechboy.com/raspberry-pi/raspberry-pi-lesson-28-controlling-a-servo-on-raspberry-pi-with-python/

http://raspberrypi.stackexchange.com/questions/24766/raspberry-pi-servo-motor-angle-calculation

http://platea.pntic.mec.es/vgonzale/cyr_0204/ctrl_rob/robotica/sistema/motores_servo.htm

http://www.nociones.de/controlar-un-servo-con-rasperry-pi-usando-rpio-pwm-y-dma/

http://www.pythonforbeginners.com/system/python-sys-argv

http://magmax.org/blog/programarsinifs/

http://electronics.stackexchange.com/questions/38125/l298-output-voltage-is-too-low

http://forums.parallax.com/discussion/156410/how-to-use-a-l298n-dual-h-bridge-with-a-microcontroller-quickstart-board

http://www.instructables.com/id/Raspberry-PI-L298N-Dual-H-Bridge-DC-Motor/

https://www.youtube.com/watch?v=W7cV9_W12sM

https://www.youtube.com/watch?v=rESbe0dGbls

http://computers.tutsplus.com/tutorials/controlling-dc-motors-using-python-with-a-raspberry-pi--cms-20051

http://electronilab.co/tienda/modulo-lm2596-convertidor-de-voltaje-dc-dc-buck-1-25v-35v/

http://www.inventable.eu/2013/09/16/7-mini-circuitos/

https://sites.google.com/site/picuino/pid_controller#TOC-Acci-n-de-control-Proporcional

http://scontrol2.blogspot.com.es/

http://scontrol2.blogspot.com.es/2007/12/controladores-pid.html

https://es.scribd.com/doc/2634725/CONTROLADORES

http://e-ducativa.catedu.es/44700165/aula/archivos/repositorio//4750/4926/html/14_controlador_de_accin_proporcional_y_derivativa_pd.html

http://www.alvarohurtado.es/threads-en-python/

http://www.genbetadev.com/python/multiprocesamiento-en-python-threads-a-fondo-introduccion

http://stackoverflow.com/questions/25676835/signal-handling-in-multi-threaded-python

https://docs.python.org/3/library/threading.html

ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

SECCIÓN DE INGENIERÍA INFORMÁTICA

Anexos

Titulación: Ingeniería en Informática

Alumno: Francisco Raúl Machín Castilla

Tutor: Santiago Torres Álvarez

Julio 2016

Historia de Raspberry Pi

Raspberry Pi fue ideado en 2006 pero no fue lanzado al mercado hasta febrero de 2012. Fue desarrollao por un grupo de la Universidad de Cambridge y su misión es fomentar la enseñanza de las ciencias de la computación en los niños. De hecho, en enero de este año, Google donó más de 15.000 unidades para los colegios del Reino Unido. Raspberry Pi es una excelente herramienta para para aprender electrónica y programación, de hecho, los primeros diseños, con sus esquemas y el circuito impreso están disponibles para su descarga pública.

En Mayo de 2009, la fundación Raspberry Pi fue creada en Caldecote, South Cambridgeshire, en Reino Unido, como una asociación caricativa que es regulada por la Comisión de Caridad de Inglaterra y Gales. “La Fundación Raspberry Pi” surge con el objetivo de desarrollar el uso y entendimiento de los ordenadores en los niños. Inculcando estos valores aseguran estimular e incentivar el interés de los ordenadores en las primeras generaciones, ayudando a los niños a utilizarlos sin miedo, abriendo su mentalidad y educándolos en la ética del “ábrelo y mira cómo funciona”. No es en vano que, el ideólogo del proyecto Raspberry Pi, David Braven, sea un antiguo desarrollador de videojuegos. Afirma que su objetivo sea que los niños lleguen a comprender un el funcionamiento básico de un ordenador de forma divertida y sean los niños, aquellos que desarrollen y amplíen sus dispositivos. El co-fundador de la fundación es Eben Christopher Upton, director técnico y Arquitecto de Sistemas de Circuitos integrados de aplicaciones específicas para la empresa Broadcom Corporation que es uno de los principales fabricantes de circuitos integrados para comunicaciones de banda ancha de los Estados Unidos.

Figura 1: Eben Upton, creador de Raspberry Pi

Eben Upton, se puso en contacto con un grupo de profesores, académicos y entusiastas de la informática para crear un ordenador con la intención de animar a los niños en el mundo de la informática tal y como lo hizo en 1981 con el ordenador Acorn BBC Micro.

Figura 2: Computador BBC Micro (Acorn Proton)

La BBC quería basar su proyecto en un microcomputador capaz de realizar varias tareas que ellos podían entonces demostrar en su serie de TV *The Computer Programme* (El Programa de Computadora) (1981). La lista de temas incluía programación, gráficos, sonido y música, Teletexto, controlando hardware externo, inteligencia artificial, etc. La BBC decidió distinguir un microcomputador, después elaboró una especificación bastante ambiciosa (para su tiempo) y solicitó ofertas.

El primer prototipo de Raspberry Pi basado en arquitectura ARM fue montado en un paquete del mismo tamaño que una memoria USB. Tenía un solo puerto de USB en un extremo y un puerto HDMI en el otro.

La Arquitectura ARM, desarrollada por Acorn Computers a finales de los 80, es relativamente poco conocida en el mundo de las computadoras de escritorios, los actuales PC. Donde verdaderamente destaca la arquitectura ARM es en los dispositivos móviles, el teléfono de nuestro bolsillo que usamos habitualmente está sostenido en al menos, un núcleo de procesacimiento basado en la arquitectura ARM.

En agosto de 2011, se fabricaron cincuenta placas Alpha del modelo inicial, el modelo A de Raspberry Pi. En diciembre de 2011, se ensamblaron 25 placas Beta, modelo B de Raspberry Pi. Durante la primera semana de diciembre de 2011, se pusieron a la subasta diez placas en Ebay. Una nota curiosa fue que, debido al anticipado anuncio de poner a la venta las placas, a final de febrero de 2012, la fundación sufrió un colapso en sus servidores web debido a los refrescos de páginas web desde los navegadores de la gente que estaba interesada en comprar la placa.

La primera tanda de tarjetas fue un lote de 10.000 placas fabricadas en Taiwány China, en lugar del Reino Unido. El motivo era obvio: un abaratamiento de los costes de producción y de acortar el plazo de entrega del producto ya que los fabricantes chinos ofrecían un plazo de entrega de 4 semanas y en el Reino Unido el plazo de entrega se triplicaba hasta un total de 12 semanas. La fundación tras este ahorro podría invertir más dinero en investigación y desarrollo.

Las primeras ventas empezaron el 29 de febrero de 2012 (Modelo B) a través de las dos tiendas que vendían las placas en ese momento, Premier Farnell y RS Components. Ambas compañías cuyos orígenes pertenecen al Reino Unido y que suministran componentes eléctricos y electrónicos, la primera fundada el 1939 en Leeds y la segunda dos años antes en Londres. En los seis meses siguientes llegarían a vender hasta 500.000 unidades.

Desde el 16 de abril de 2012 los primeros compradores ya empezaron a informar que habían recibido su Raspberry Pi, el 22 de mayo del mismo año ya superaban las ventas en más 20.000 unidades.

Por diversos motivos, la fundación ya planteaba su traslado de la producción de las placas de vuelta al Reino Unido, más concretamente a una fábrica de Sony a condición que en ella se produjeran alrededor de 30.000 unidades cada mes y con ellos crearían algo más de 30 puestos de trabajo.

El 4 de septiembre de 2013, la fundación decide lanzar el modelo A de Raspberry Pi con unas características más reducidas, 256Mb de RAM y sin puerto ethernet pero con un precio más asequible que el modelo B.

SISTEMAS DE CONTROL

Control Proporcional – Derivativo

También el regulador derivativo lleva asociado con frecuencia la actuación de un regulador proporcional. La salida del controlador en respuesta a una señal de entrada se representa en forma de rampa unitaria y no tiene sentido representarla como una señal de error tipo escalón.

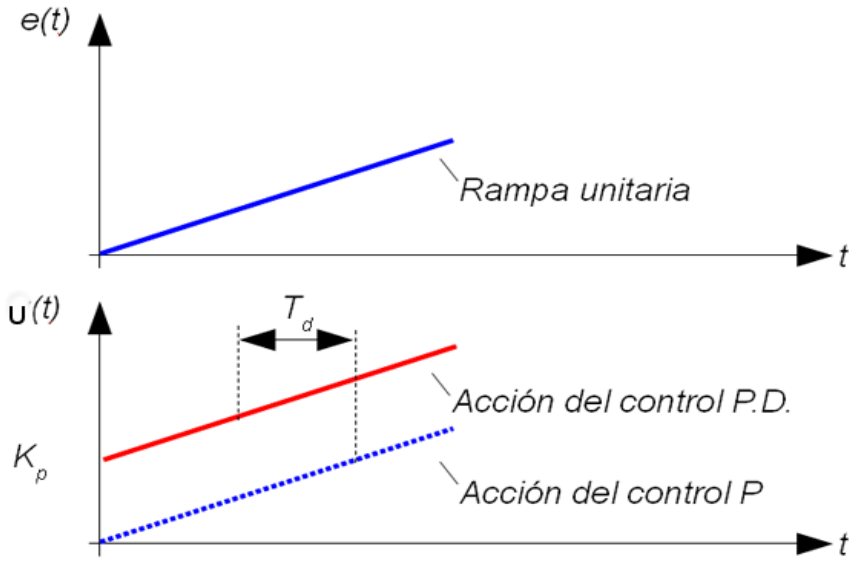


Ilustración 1: Grafica comportamiento sistema control PD

La ventaja de este tipo de controlador es que aumenta la velocidad de respuesta del sistema de control. Al actuar en conjunto con una acción proporcional, las características de un controlador derivativo, provocan una apreciable mejora de la velocidad de respuesta del sistema, aunque pierde precisión en la salida mientras que funciona el control derivativo.

Matemáticamente tenemos:

$$u(t) = K_p e(t) + K_p T_d \frac{de(t)}{dt}$$

donde Kp es la ganancia proporcional y Td es una constante denominada tiempo derivativo.

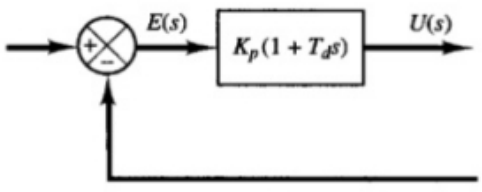


Ilustración 2: Diagrama de bloques del controlador PD

Como desventaja, la acción derivativa amplifica las señales de ruido y puede provocar un efecto de saturación en el actuador.

Control Proporcional – Integral - Derivativo

De igual forma que los anteriores sistemas de control pueden combinarse, también podemos aprovechar cada una de las ventajas básicas de cada uno de ellos y de esta forma, si la señal de error varía lentamente en el tiempo, entonces *predomina la acción integral* y si, por el contrario, la señal de error varía muy rápido en el tiempo, predomina *la acción derivativa.*

Matemáticamente, podríamos ver el comportamiento de un controlador PID reflejado en la siguiente fórmula:

$$u(t)=Kpe(t)+Kpt_d\cdot\frac{det}{dt}+Kp\frac{1}{t_i}\int e(t)dt$$

y su función de transferencia como

$$\frac{U(S)}{E(S)}=Kp\left(a+T_d\cdot s+\frac{1}{(T_i\cdot s)}\right)$$

En cuanto al comportamiento del sistema de control Proporcional – Integral – Derivado podemos observarlo en la siguiente gráfica:

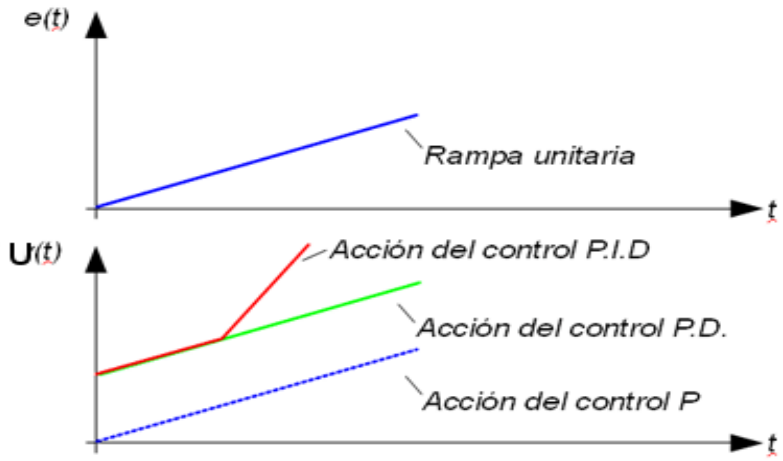


Ilustración 3: Gráfica comportamiento sistema de control PID

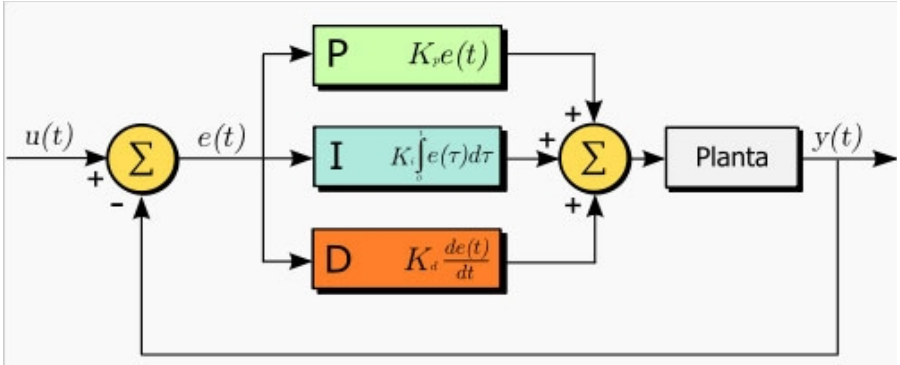


Ilustración 4: Diagrama de bloques del controlador PID

Un ejemplo donde podríamos entender el comportamiento de un sistema de control PID podría tratarse de la conducción de un automóvil:

Imaginemos que el cerebro (controlador) de un conductor da la orden de cambio de dirección y velocidad a las manos y los pies (actuadores) mientras conduce realizando una determinada maniobra. Supongamos que la maniobra es una situación normal de conducción de manera que el conductor modificará su dirección hasta la deseada con más o menos precisión. Aquí entra en control proporcional y justo cuando la dirección esté próxima al valor deseado, entra en juego el control integral que, reducirá el posible error debido al control proporcional hasta posicionar el volante en el punto preciso. **Ésta acción es la que necesitamos en el robot prototipo del proyecto que le ayudará a llegar al lugar que desea llegar.** Si por el contrario, la maniobra de conducción precisa de un giro brusco entonces el control derivativo adquiere más importancia, aumentando la velocidad de respuesta del sistema para luego entrar de nuevo en juego el control proporcional y finalmente el integral, cuando llegamos a la dirección deseada. Hay que tener en cuenta que si aplicamos la acción derivativa, conseguiremos mayor velocidad de respuesta pero a costa de perder precisión en la maniobra.

```
import threading, time, sys, signal, math, os, RPIO
import RPi.GPIO as GPIO
from RPIO import PWM

GPIO.setmode(GPIO.BCM)
RPIO.setup(17, RPIO.IN)
RPIO.setup(17, RPIO.IN, pull_up_down=RPIO.PUD_UP)
RPIO.setup(22, RPIO.IN)
RPIO.setup(22, RPIO.IN, pull_up_down=RPIO.PUD_UP)
RPIO.setwarnings(False)
RPIO.setup(24, RPIO.IN)
RPIO.setup(24, RPIO.IN, pull_up_down=RPIO.PUD_UP)
RPIO.setup(25, RPIO.IN)
RPIO.setup(25, RPIO.IN, pull_up_down=RPIO.PUD_UP)

running = True


# Funcion que activa los servos correspondientes para recoger y soltar las pelotas de tenis
def accion_servo(servo_mover):

        servo = PWM.Servo()
        #activamos servo delantero para recoger pelotas (sube 90 - 40 grados evitar choque varilla delant
era suelo)
        if (servo_mover == 'D'):
                servo.set_servo(27, 2500)
                time.sleep(2)
                servo.set_servo(27, 1100)
                time.sleep(1)
                servo.stop_servo(27)
                RPIO.setup(27, RPIO.IN)
        #activamos servo trasero para soltar las pelotas ( 54 - 4 grados aprox)
        else:
                servo.set_servo(23, 1500)
                time.sleep(1)
                servo.set_servo(23, 100)
                time.sleep(1)
                servo.stop_servo(23)
                RPIO.setup(23, RPIO.IN)

# Funcion que hace girar los motores hacia adelante
def adelante():

        global contador1
        global contador2

        anterior_izq_A=RPIO.input(17)
        anterior_izq_B=RPIO.input(22)

        anterior_der_A=RPIO.input(24)
        anterior_der_B=RPIO.input(25)

        while running:

                input_value_17 = RPIO.input(17)
                input_value_22 = RPIO.input(22)


                if (input_value_17 != anterior_izq_A or input_value_22 != anterior_izq_B):
                        # IZQ = 0  DER = 0
                        if (anterior_izq_A == False and anterior_izq_B == False):
                                # SIG_IZQ = 1 SIG_DER = 0
                                if (input_value_17 == True and input_value_22 == False):
                                        contador1 = contador1 + 1

                                # SIG_IZQ = 1 SIG_DER = 1
                                elif (input_value_17 == True and input_value_22 == True):
                                        contador1 = contador1 + 2

                                # SIG_IZQ = 0 SIG_DER = 1
                                elif (input_value_17 == False and input_value_22 == True):
                                        contador1 = contador1 + 3

                        # IZQ = 0    DER = 1
                        elif (anterior_izq_A== False and anterior_izq_B == True):

                                # SIG_IZQ = 0 SIG_DER = 0
                                if (input_value_17 == False and input_value_22 == False):
                                        contador1 = contador1 + 1

                                # SIG_IZQ = 1 SIG_DER = 0
                                elif (input_value_17 == True and input_value_22 == False):
                                        contador1 = contador1 + 2

                                # SIG_IZQ = 1 SIG_DER = 1
                                elif (input_value_17 == True and input_value_22 == True):
```

```
                                contador1 = contador1 + 3

                        # IZQ = 1    DER = 0
                        elif (anterior_izq_A== True and anterior_izq_B == False):

                                # SIG_IZQ = 1 SIG_DER = 1
                                if (input_value_17 == True and input_value_22 == True):
                                        contador1 = contador1 + 1

                                # SIG_IZQ = 0 SIG_DER = 1
                                elif (input_value_17 == False  and input_value_22 == True):
                                        contador1 = contador1 + 2

                                # SIG_IZQ = 0 SIG_DER = 0
                                elif (input_value_17 == False and input_value_22 == False):
                                        contador1 = contador1 + 3

                        # IZQ = 1    DER = 1
                        elif (anterior_izq_A== True and anterior_izq_B == True):

                                # SIG_IZQ = 0 SIG_DER = 1
                                if (input_value_17 == False and input_value_22 == True):
                                        contador1 = contador1 + 1

                                # SIG_IZQ = 0 SIG_DER = 0
                                elif (input_value_17 == False  and input_value_22 == False):
                                        contador1 = contador1 + 2

                                # SIG_IZQ = 1 SIG_DER = 0
                                elif (input_value_17 == True and input_value_22 == False):
                                        contador1 = contador1 + 3

                anterior_izq_A = input_value_17
                anterior_izq_B = input_value_22

                # MOTOR DERECHO

                input_value_24 = RPIO.input(24)
                input_value_25 = RPIO.input(25)

                if (input_value_24 != anterior_der_A or input_value_25 != anterior_der_B):
                        #  IZQ = 0  DER = 0
                        if (anterior_der_A == False and anterior_der_B == False):

                                # SIG_IZQ = 1 SIG_DER = 0
                                if (input_value_24 == True and input_value_25 == False):
                                        contador2 = contador2 + 1

                                # SIG_IZQ = 1 SIG_DER = 1
                                elif (input_value_24 == True and input_value_25 == True):
                                        contador2 = contador2 + 2

                                # SIG_IZQ = 0 SIG_DER = 1
                                elif (input_value_24 == False and input_value_25 == True):
                                        contador2 = contador2 + 3

                        # IZQ = 0    DER = 1
                        elif (anterior_der_A== False and anterior_der_B == True):

                                # SIG_IZQ = 0 SIG_DER = 1
                                if (input_value_24 == False and input_value_25 == False):
                                        contador2 = contador2 + 1

                                # SIG_IZQ = 1 SIG_DER = 0
                                elif (input_value_24 == True and input_value_25 == False):
                                        contador2 = contador2 + 2

                                # SIG_IZQ = 1 SIG_DER = 1
                                elif (input_value_24 == True and input_value_25 == True):
                                        contador2 = contador2 + 3

                        # IZQ = 1    DER = 0
                        elif (anterior_der_A== True and anterior_der_B == False):

                                # SIG_IZQ = 1 SIG_DER = 1
                                if (input_value_24 == True and input_value_25 == True):
                                        contador2 = contador2 + 1

                                # SIG_IZQ = 0 SIG_DER = 1
                                elif (input_value_24 == False  and input_value_25 == True):
                                        contador2 = contador2 + 2

                                # SIG_IZQ = 0 SIG_DER = 0
                                elif (input_value_24 == False and input_value_25 == False):
                                        contador2 = contador2 + 3
```

```
                        # IZQ = 1    DER = 1
                        elif (anterior_der_A== True and anterior_der_B == True):

                                # SIG_IZQ = 0 SIG_DER = 1
                                if (input_value_24 == False and input_value_25 == True):
                                        contador2 = contador2 + 1

                                # SIG_IZQ = 0 SIG_DER = 0
                                elif (input_value_24 == False  and input_value_25 == False):
                                        contador2 = contador2 + 2

                                # SIG_IZQ = 1 SIG_DER = 0
                                elif (input_value_24 == True and input_value_25 == False):
                                        contador2 = contador2 + 3


                anterior_der_A = input_value_24
                anterior_der_B = input_value_25

        #END_WHILE
        hilo.stopped = True
        return([contador1, contador2])


# Funcion que hace girar los motores hacia detras
def atras():

        global contador1
        global contador2

        anterior_izq_A=RPIO.input(17)
        anterior_izq_B=RPIO.input(22)

        anterior_der_A=RPIO.input(24)
        anterior_der_B=RPIO.input(25)

        while running:

                input_value_17 = RPIO.input(17)
                input_value_22 = RPIO.input(22)

                if (input_value_17 != anterior_izq_A or input_value_22 != anterior_izq_B):

                        # IZQ = 0  DER = 0
                        if (anterior_izq_A == False and anterior_izq_B == False):

                                # SIG_IZQ = 1 SIG_DER = 0
                                if (input_value_17 == True and input_value_22 == False):
                                        contador1 = contador1 - 3

                                # SIG_IZQ = 1 SIG_DER = 1
                                elif (input_value_17 == True and input_value_22 == True):
                                        contador1 = contador1 - 2

                                # SIG_IZQ = 0 SIG_DER =
                                elif (input_value_17 == False and input_value_22 == True):
                                        contador1 = contador1 - 1

                        # IZQ = 0    DER = 1
                        elif (anterior_izq_A== False and anterior_izq_B == True):

                                # SIG_IZQ = 0 SIG_DER = 0
                                if (input_value_17 == False and input_value_22 == False):
                                        contador1 = contador1 - 3

                                # SIG_IZQ = 1 SIG_DER = 0
                                elif (input_value_17 == True and input_value_22 == False):
                                        contador1 = contador1 - 2

                                # SIG_IZQ = 1 SIG_DER = 1
                                elif (input_value_17 == True and input_value_22 == True):
                                        contador1 = contador1 - 1

                        # IZQ = 1    DER = 0
                        elif (anterior_izq_A== True and anterior_izq_B == False):

                                # SIG_IZQ = 1 SIG_DER = 1
                                if (input_value_17 == True and input_value_22 == True):
                                        contador1 = contador1 - 3

                                # SIG_IZQ = 0 SIG_DER = 1
                                elif (input_value_17 == False  and input_value_22 == True):
                                        contador1 = contador1 - 2

                                # SIG_IZQ = 0 SIG_DER = 0
```

```
                        elif (input_value_17 == False and input_value_22 == False):
                                contador1 = contador1 - 1

                # IZQ = 1    DER = 1
                elif (anterior_izq_A== True and anterior_izq_B == True):

                        # SIG_IZQ = 0 SIG_DER = 1
                        if (input_value_17 == False and input_value_22 == True):
                                contador1 = contador1 - 3

                        # SIG_IZQ = 0 SIG_DER = 0
                        elif (input_value_17 == False  and input_value_22 == False):
                                contador1 = contador1 - 2

                        # SIG_IZQ = 1 SIG_DER = 0
                        elif (input_value_17 == True and input_value_22 == False):
                                contador1 = contador1 - 1

        anterior_izq_A = input_value_17
        anterior_izq_B = input_value_22

        # MOTOR DERECHO

        input_value_24 = RPIO.input(24)
        input_value_25 = RPIO.input(25)

        if (input_value_24 != anterior_der_A or input_value_25 != anterior_der_B):
                #  IZQ = 0  DER = 0
                if (anterior_der_A == False and anterior_der_B == False):

                        # SIG_IZQ = 1 SIG_DER = 0
                        if (input_value_24 == True and input_value_25 == False):
                                contador2 = contador2 - 3

                        # SIG_IZQ = 1 SIG_DER = 1
                        elif (input_value_24 == True and input_value_25 == True):
                                contador2 = contador2 - 2

                        # SIG_IZQ = 0 SIG_DER = 1
                        elif (input_value_24 == False and input_value_25 == True):
                                contador2 = contador2 - 1

                # IZQ = 0    DER = 1
                elif (anterior_der_A== False and anterior_der_B == True):

                        # SIG_IZQ = 0 SIG_DER = 1
                        if (input_value_24 == False and input_value_25 == False):
                                contador2 = contador2 - 3

                        # SIG_IZQ = 1 SIG_DER = 0
                        elif (input_value_24 == True and input_value_25 == False):
                                contador2 = contador2 - 2

                        # SIG_IZQ = 1 SIG_DER = 1
                        elif (input_value_24 == True and input_value_25 == True):
                                contador2 = contador2 - 1

                # IZQ = 1    DER = 0
                elif (anterior_der_A== True and anterior_der_B == False):

                        # SIG_IZQ = 1 SIG_DER = 1
                        if (input_value_24 == True and input_value_25 == True):
                                contador2 = contador2 - 3

                        # SIG_IZQ = 0 SIG_DER = 1
                        elif (input_value_24 == False  and input_value_25 == True):
                                contador2 = contador2 - 2

                        # SIG_IZQ = 0 SIG_DER = 0
                        elif (input_value_24 == False and input_value_25 == False):
                                contador2 = contador2 - 1

                # IZQ = 1    DER = 1
                elif (anterior_der_A== True and anterior_der_B == True):

                        # SIG_IZQ = 0 SIG_DER = 1
                        if (input_value_24 == False and input_value_25 == True):
                                contador2 = contador2 - 3

                        # SIG_IZQ = 0 SIG_DER = 0
                        elif (input_value_24 == False  and input_value_25 == False):
                                contador2 = contador2 - 2

                        # SIG_IZQ = 1 SIG_DER = 0
                        elif (input_value_24 == True and input_value_25 == False):
                                contador2 = contador2 - 1
```

```
                anterior_der_A = input_value_24
                anterior_der_B = input_value_25

        #END_WHILE
        hilo.stopped = True
        return([contador1,contador2])


# Funcion que realiza el giro del robot
def giro():


        global contador1
        global contador2
        global sentido


        if (sentido == 'IZQUIERDA'):

                #ENCODER IZQUIERDO
                anterior_izq_A=RPIO.input(17)
                anterior_izq_B=RPIO.input(22)

                #ENCODER DERECHO
                anterior_der_A=RPIO.input(24)
                anterior_der_B=RPIO.input(25)


                while running:
                        # MOTOR IZQUIERDO ATRAS
                        input_value_17 = RPIO.input(17)
                        input_value_22 = RPIO.input(22)

                        if (input_value_17 != anterior_izq_A or input_value_22 != anterior_izq_B):

                                # IZQ = 0  DER = 0
                                if (anterior_izq_A == False and anterior_izq_B == False):

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        if (input_value_17 == True and input_value_22 == False):
                                                contador1 = contador1 - 3

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        elif (input_value_17 == True and input_value_22 == True):
                                                contador1 = contador1 - 2

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        elif (input_value_17 == False and input_value_22 == True):
                                                contador1 = contador1 - 1

                                # IZQ = 0    DER = 1
                                elif (anterior_izq_A== False and anterior_izq_B == True):

                                                # SIG_IZQ = 0 SIG_DER = 0
                                        if (input_value_17 == False and input_value_22 == False):
                                                contador1 = contador1 - 3

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        elif (input_value_17 == True and input_value_22 == False):
                                                contador1 = contador1 - 2

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        elif (input_value_17 == True and input_value_22 == True):
                                                contador1 = contador1 - 1

                                # IZQ = 1    DER = 0
                                elif (anterior_izq_A== True and anterior_izq_B == False):

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        if (input_value_17 == True and input_value_22 == True):
                                                contador1 = contador1 - 3

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        elif (input_value_17 == False  and input_value_22 == True):
                                                contador1 = contador1 - 2

                                        # SIG_IZQ = 0 SIG_DER = 0
                                        elif (input_value_17 == False and input_value_22 == False):
                                                contador1 = contador1 - 1

                                # IZQ = 1    DER = 1
                                elif (anterior_izq_A== True and anterior_izq_B == True):

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        if (input_value_17 == False and input_value_22 == True):
```

```
                                        contador1 = contador1 - 3

                                # SIG_IZQ = 0 SIG_DER = 0
                                elif (input_value_17 == False  and input_value_22 == False):
                                        contador1 = contador1 - 2

                                # SIG_IZQ = 1 SIG_DER = 0
                                elif (input_value_17 == True and input_value_22 == False):
                                        contador1 = contador1 - 1

                        anterior_izq_A = input_value_17
                        anterior_izq_B = input_value_22


                        # MOTOR DERECHO ADELANTE
                        input_value_24 = RPIO.input(24)
                        input_value_25 = RPIO.input(25)

                        if (input_value_24 != anterior_der_A or input_value_25 != anterior_der_B):
                                #  IZQ = 0  DER = 0
                                if (anterior_der_A == False and anterior_der_B == False):

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        if (input_value_24 == True and input_value_25 == False):
                                                contador2 = contador2 + 1

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        elif (input_value_24 == True and input_value_25 == True):
                                                contador2 = contador2 + 2

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        elif (input_value_24 == False and input_value_25 == True):
                                                contador2 = contador2 + 3

                                # IZQ = 0    DER = 1
                                elif (anterior_der_A== False and anterior_der_B == True):

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        if (input_value_24 == False and input_value_25 == False):
                                                contador2 = contador2 + 1

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        elif (input_value_24 == True and input_value_25 == False):
                                                contador2 = contador2 + 2

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        elif (input_value_24 == True and input_value_25 == True):
                                                contador2 = contador2 + 3

                                # IZQ = 1    DER = 0
                                elif (anterior_der_A== True and anterior_der_B == False):

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        if (input_value_24 == True and input_value_25 == True):
                                                contador2 = contador2 + 1

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        elif (input_value_24 == False  and input_value_25 == True):
                                                contador2 = contador2 + 2

                                        # SIG_IZQ = 0 SIG_DER = 0
                                        elif (input_value_24 == False and input_value_25 == False):
                                                contador2 = contador2 + 3

                                # IZQ = 1    DER = 1
                                elif (anterior_der_A== True and anterior_der_B == True):

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        if (input_value_24 == False and input_value_25 == True):
                                                contador2 = contador2 + 1

                                        # SIG_IZQ = 0 SIG_DER = 0
                                        elif (input_value_24 == False  and input_value_25 == False):
                                                contador2 = contador2 + 2

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        elif (input_value_24 == True and input_value_25 == False):
                                                contador2 = contador2 + 3

                        anterior_der_A = input_value_24
                        anterior_der_B = input_value_25

                #END_WHILE

                hilo.stopped = True
```

```
        #END_IF


        #Giramos a la derecha (IZQ-Atras, DER-Adelante)
        else:

                anterior_izq_A=RPIO.input(17)
                anterior_izq_B=RPIO.input(22)

                anterior_der_A=RPIO.input(24)
                anterior_der_B=RPIO.input(25)

                while running:
                        # MOTOR IZQUIERDO ADELANTE
                        input_value_17 = RPIO.input(17)
                        input_value_22 = RPIO.input(22)

                        if (input_value_17 != anterior_izq_A or input_value_22 != anterior_izq_B):

                                # IZQ = 0  DER = 0
                                if (anterior_izq_A == False and anterior_izq_B == False):

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        if (input_value_17 == True and input_value_22 == False):
                                                contador1 = contador1 + 1

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        elif (input_value_17 == True and input_value_22 == True):
                                                contador1 = contador1 + 2

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        elif (input_value_17 == False and input_value_22 == True):
                                                contador1 = contador1 + 3

                                # IZQ = 0    DER = 1
                                elif (anterior_izq_A== False and anterior_izq_B == True):

                                        # SIG_IZQ = 0 SIG_DER = 0
                                        if (input_value_17 == False and input_value_22 == False):
                                                contador1 = contador1 + 1

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        elif (input_value_17 == True and input_value_22 == False):
                                                contador1 = contador1 + 2

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        elif (input_value_17 == True and input_value_22 == True):
                                                contador1 = contador1 + 3

                                # IZQ = 1    DER = 0
                                elif (anterior_izq_A== True and anterior_izq_B == False):

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        if (input_value_17 == True and input_value_22 == True):
                                                contador1 = contador1 + 1

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        elif (input_value_17 == False  and input_value_22 == True):
                                                contador1 = contador1 + 2

                                        # SIG_IZQ = 0 SIG_DER = 0
                                        elif (input_value_17 == False and input_value_22 == False):
                                                contador1 = contador1 + 3

                                # IZQ = 1    DER = 1
                                elif (anterior_izq_A== True and anterior_izq_B == True):

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        if (input_value_17 == False and input_value_22 == True):
                                                contador1 = contador1 + 1

                                        # SIG_IZQ = 0 SIG_DER = 0
                                        elif (input_value_17 == False  and input_value_22 == False):
                                                contador1 = contador1 + 2

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        elif (input_value_17 == True and input_value_22 == False):
                                                contador1 = contador1 + 3

                        anterior_izq_A = input_value_17
                        anterior_izq_B = input_value_22


                        # MOTOR DERECHO ATRAS
```

```
                        input_value_24 = RPIO.input(24)
                        input_value_25 = RPIO.input(25)

                        if (input_value_24 != anterior_der_A or input_value_25 != anterior_der_B):
                                #  IZQ = 0  DER = 0
                                if (anterior_der_A == False and anterior_der_B == False):

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        if (input_value_24 == True and input_value_25 == False):
                                                contador2 = contador2 - 3

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        elif (input_value_24 == True and input_value_25 == True):
                                                contador2 = contador2 - 2

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        elif (input_value_24 == False and input_value_25 == True):
                                                contador2 = contador2 - 1

                                # IZQ = 0    DER = 1
                                elif (anterior_der_A== False and anterior_der_B == True):

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        if (input_value_24 == False and input_value_25 == False):
                                                contador2 = contador2 - 3

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        elif (input_value_24 == True and input_value_25 == False):
                                                contador2 = contador2 - 2

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        elif (input_value_24 == True and input_value_25 == True):
                                                contador2 = contador2 - 1

                                # IZQ = 1    DER = 0
                                elif (anterior_der_A== True and anterior_der_B == False):

                                        # SIG_IZQ = 1 SIG_DER = 1
                                        if (input_value_24 == True and input_value_25 == True):
                                                contador2 = contador2 - 3

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        elif (input_value_24 == False  and input_value_25 == True):
                                                contador2 = contador2 - 2

                                        # SIG_IZQ = 0 SIG_DER = 0
                                        elif (input_value_24 == False and input_value_25 == False):
                                                contador2 = contador2 - 1

                                # IZQ = 1    DER = 1
                                elif (anterior_der_A== True and anterior_der_B == True):

                                        # SIG_IZQ = 0 SIG_DER = 1
                                        if (input_value_24 == False and input_value_25 == True):
                                                contador2 = contador2 - 3

                                        # SIG_IZQ = 0 SIG_DER = 0
                                        elif (input_value_24 == False  and input_value_25 == False):
                                                contador2 = contador2 - 2

                                        # SIG_IZQ = 1 SIG_DER = 0
                                        elif (input_value_24 == True and input_value_25 == False):
                                                contador2 = contador2 - 1

                        anterior_der_A = input_value_24
                        anterior_der_B = input_value_25


                #END_WHILE
                hilo.stopped = True

        #END_ELSE

        return ([contador1, contador2])

# HILO encargado de realizar el sistema de control proporcional integral, calcular el numero de ticks
def monitor(tid, itemID=None, threshold=None):
        global running
        global sentido

        # NUMERO DE TICKS.
        centimetros = int(sys.argv[2])
        if (sentido == "ADELANTE") or (sentido == "ATRAS"):
                num_ticks_der = (TOTAL_TICKS_VUELTA_DER * centimetros)/dos_pi_R
                num_ticks_izq = ((TOTAL_TICKS_VUELTA_IZQ + tolerancia) * centimetros)/dos_pi_R
                #print ("Numero de ticks izq es -> %s") % num_ticks_izq
```

```
        #print ("Numero de ticks der es -> %s") % num_ticks_der
#SI GIRA DERECHA o IZQUIERDA
elif (sentido == "DERECHA") or (sentido == "IZQUIERDA"):
        num_ticks_izq = centimetros * TICKS_POR_GRADO_IZQ * 2
        num_ticks_der = centimetros * TICKS_POR_GRADO_DER * 2
        num_ticks_izq = (num_ticks_izq )   + tolerancia
        num_ticks_der = (num_ticks_der )

cont_aux = 0

tole_error = 2
T = 0.05

ukm1_der = 0.0
ukm1_izq = 0.0
ekm1_d = 0.0
ekm1_i = 0.0

#VALORES A KsuP y KsuI para conseguir movimiento
kp = 0.75
ki = 0.01 * kp

p0 = (ki * T/2) + kp
p1 = (ki * T/2) - kp

#print "kp =%s, ki=%s, p0=%s, p1=%s" % (kp,ki,p0,p1)

#MOTOR IZQUIERDO = Motor1
#MOTOR DERECHO = Motor2
if (sentido == "ADELANTE"):
        #print "ADELANTE"
        Motor1 = 28
        Motor2 = 29

elif (sentido == "ATRAS"):
        #print "ATRAS"
        Motor1 = 30
        Motor2 = 31

elif (sentido == "IZQUIERDA"):
        #print "GIRO_IZQUIERDA"
        Motor1 = 30
        Motor2 = 29
else:
        #print "GIRO_DERECHA"
        Motor1 = 28
        Motor2 = 31

GPIO.setup(Motor1, GPIO.OUT)
p_izq = GPIO.PWM(Motor1, 100)
p_izq.start(0)

GPIO.setup(Motor2, GPIO.OUT)
p_der = GPIO.PWM(Motor2, 100)
p_der.start(0)

freq_muestreo = T * 10000
while running:
        cont_aux = cont_aux + 1
        if (math.fmod(cont_aux, int(freq_muestreo))  == 0):

                ek_d = num_ticks_der - abs(contador2)
                uk_d = ukm1_der + p0 * ek_d + p1 * ekm1_d
                ek_i = num_ticks_izq - abs(contador1)
                uk_i = ukm1_izq + p0 * ek_i + p1 * ekm1_i

                uk_i_s = min(uk_i,100.0)
                uk_i_s = max(uk_i_s, 0.0)
                uk_d_s = min(uk_d,100.0)
                uk_d_s = max(uk_d_s, 0.0)

                uk_max = max(uk_i_s, uk_d_s)
                p_izq.ChangeDutyCycle(int(uk_max))
                p_der.ChangeDutyCycle(int(uk_max))


                #PARA EL GIRO la condicion de parada distinta
                if (sentido == "IZQUIERDA") or (sentido == "DERECHA"):
                        if (max(ek_i, ek_d) < tole_error):
                                p_der.stop()
                                p_izq.stop()
                                GPIO.cleanup()
                                break

                elif (sentido == "ADELANTE") or (sentido == "ATRAS"):
                        if (abs(ek_i) < tole_error) or (abs(ek_d) < tole_error):
```

```
                                        p_der.stop()
                                        p_izq.stop()
                                        GPIO.cleanup()
                                        break

                                #actualizamos variables
                                ukm1_der = uk_d
                                ukm1_izq = uk_i

                                ekm1_d = ek_d
                                ekm1_i = ek_i

        hilo2.stopped =True
        running = False
        #end_WHILE

#SEGUNDO HILO para actualizar los contadores llamando al movimiento detras, delante o giro
def actualizo_contador(tid, itemID=None, threshold=None):
        global sentido
        global contador1
        contador1 = 0
        global contador2
        contador2 = 0

        global PI
        PI = 3.14

        global dos_pi_R
        dos_pi_R = 34.5575

        global TOTAL_TICKS_VUELTA_IZQ
        TOTAL_TICKS_VUELTA_IZQ = 86
        global TOTAL_TICKS_VUELTA_DER
        TOTAL_TICKS_VUELTA_DER = 89

        global tolerancia
        tolerancia = abs(TOTAL_TICKS_VUELTA_DER - TOTAL_TICKS_VUELTA_IZQ)

        global TICKS_POR_GRADO_IZQ
        TICKS_POR_GRADO_IZQ = 0.3752 * 2

        global TICKS_POR_GRADO_DER
        TICKS_POR_GRADO_DER = 0.3883 * 2

        if (sys.argv[1] == 'F'):
                sentido = "ADELANTE"
                resultados = adelante()

        elif (sys.argv[1] == 'B'):
                sentido = "ATRAS"
                resultados = atras()
                #servo = 'D'
                #accion_servo(servo)

        elif (sys.argv[1] == 'L' or sys.argv[1] == 'R'):
                sentido = "IZQUIERDA"
                if (sys.argv[1] == 'R'):
                        sentido = "DERECHA"
                resultados = giro()

        RPIO.cleanup()
        print (resultados)


def _handle_signal(signal, frame):
        global running
        running = False

if __name__ == '__main__':
        signal.signal(signal.SIGTERM, _handle_signal)
        signal.signal(signal.SIGINT, _handle_signal)

hilo = threading.Thread(target=monitor, args=(1,), kwargs={'itemID':'1', 'threshold':60})
hilo2 = threading.Thread(target=actualizo_contador, args=(1,), kwargs={'itemID':'2', 'threshold':60})
hilo2.start()
hilo.start()
hilo.join(30)
hilo2.join(30)
```