ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA
Grado en Ingeniería Electrónica Industrial y Automática

Trabajo Fin de Grado

Prototipo de manipulador antropomórfico para tareas colaborativas

Autores: Jesús Enrique Melián Luis
Jorge Yanes Marcelino
Tutor: Leopoldo Acosta Sánchez
Co-tutor: Antonio Luis Morell González

Quisiera agradecer a todas las personas que directa o indirectamente me han ayudado a superar este reto que culmina con el TFG.

A mis padres, hermana, abuelo y familiares, por su incondicional apoyo durante todos estos años de trabajo y lucha continua, valores que me han enseñado desde pequeño cada uno de ellos.

A Iris por estar siempre, en las buenas y en las no tanto. Llegó para quedarse a mi lado a partir del segundo año y hacerme muy feliz. Te quiero muchísimo.

A mis 'hijos' podencos, Tango y Drafi, que son unos amores. Ellos están deseando que vuelva a casa para nuestro paseo y muchas son las veces que el sentimiento ha sido mutuo, ya que ayuda a desconectar.

A los profesores de los que he aprendido, que no son todos pero si son muchos y muy buenos. Profesores con un gran nivel en sus especializaciones a lo que añaden el buen comunicar y hacerse entender. Mención especial para: Alejandro Ayala, Germán González, José Sigut, Leopoldo Acosta, Marta Sigut, Oswaldo González, Santiago Torres, Sergio Rodríguez, Severiano González, Teresa Bermúdez y Vicente Romero.

A mis compañeros por todas las experiencias vividas y por el buen ambiente de trabajo, esencial cuando se pasa tanto tiempo junto. De ellos debo destacar a: Javier León, un grande capaz de resolver cualquier cosa y seguir siendo pura humildad y bondad; Jorge Yanes, ¡ejem! ¡ejem!; Laura Arteaga, siempre puedo contar con ella; Nazaret E. Miranda, la conocí en segundo y ¡qué pena no haberla conocido antes!

A nuestros tutores, Antonio Morell y Leopoldo Acosta, por su ayuda, paciencia, inestimable confianza y colaboración en todo momento. Agradezco a Antonio el haber compartido sus conocimientos con nosotros y darnos apoyo, aún penalizando en tiempo sus propias ocupaciones. A Leo le doy gracias por tendernos siempre la mano, incluso cuando nos hemos caído él ha puesto de su parte para que podamos levantarnos y seguir adelante.

Por último, a mi compañero de TFG, Jorge Yanes, que es más que un amigo, es como mi hermano pequeño. Con él he pasado todos estos años y siempre ha estado ahí para todo lo que necesitase. Nos hemos apoyado mutuamente y cuando uno flaqueaba ahí estaba el otro para aguantarlo. Han sido muchas horas juntos, tantas que hubo momentos en los que pasaba la mayoría de mi tiempo con él. Siempre le bromeo con que tengo unas ganas terribles de acabar para perderle de vista, pero él sabe que aunque sea un poquito le echaré de menos.

Jesús Enrique Melián Luis

Desde mi punto de vista, los caminos no tienen fin. Siempre existirá un paso más, no habrá límite. Esto siempre es una oportunidad de progreso. No en comparación con el progreso de otras personas, sino con el progreso de uno mismo. Pero los caminos no son siempre rectos y planos, tienen curvas y desniveles. Es entonces cuando hay dos grandes factores que nos ayudarán a continuar, a dar cada paso. Uno es la voluntad, la determinación, la perseverancia, el coraje, etc. que tengamos para afrontar cada situación. El otro de los factores es tener a personas de gran corazón a tu lado que te ayuden de la mejor manera posible, estando contigo, y que te brinden los mejores de sus sentimientos. Por todo esto, gracias.

En primer lugar, quiero agradecer a todos los profesores del grado, que aportan siempre todos sus conocimientos para que salgamos lo mejor formados posibles. En especial, quiero agradecer a las personas que me han dado la oportunidad de poder realizar este Trabajo Fin de Grado, Leopoldo Acosta y Antonio Morell.

A Antonio lo conocí en una charla a las que nos llevó Leo sobre el Verdino, una explicación de los sistemas de alimentación, de control, de localización, un par de pruebas… Es importante el papel de investigadores como tú, pues sin investigación no puede haber futuro tecnológico. En el transcurso de este año académico, con la elaboración del proyecto, siempre has estado ahí cuando lo necesitábamos, resolviendo dudas, mostrándonos en todo momento alternativas de trabajo y enseñándonos la importancia de un trabajo continuo, el nunca darse por satisfecho a la hora de obtener resultados. Gracias.

A Leo porque desde la primera clase que tuve contigo pude sentir que no eras un docente más. No había clase en la que no te preocupases porque verdaderamente aprendiéramos. Nos enseñabas a visualizar ideas, razonarlas de una manera lógica, hacer esas ideas propias y después obtener nuevos conocimientos a partir de este punto. Siempre te preocupabas porque aprendiéramos y porque nunca nos diéramos por vencidos. Y fuera de las clases siempre has estado para ayudarme con cualquier duda que tenga, tanto académica como personal. Incluso nos diste la oportunidad de hacer este Trabajo Fin de Grado contigo. Gracias Leo.

Tampoco me puedo olvidar de mis compañeros de clase que siempre han estado ahí para hacer de la universidad un buen lugar de estudio.

Javier, contigo empezó todo. Porque gracias a ti, Javi, hemos reído, hemos disfrutado y hemos aprendido a sacar los trabajos adelante con buena cara, incluso cuando parecía que ya no quedaba tiempo. Siempre dispuesto a ayudarme, siempre dispuesto a hacer las cosas codo con codo para obtener el mejor de los resultados, la amistad.

Y otra amistad es la de Jesús. Desde el primer día, este cascarrabias ha estado ahí, en las buenas y en las no tan buenas, siempre dispuesto a ayudarme, a darme a una opinión y a comprenderme. Hemos tenido la oportunidad de realizar este Trabajo Fin de Grado juntos y, tras todas las horas pasadas en compañía el uno del otro, puedo decir que nuestra amistad ha crecido hasta el punto de que sé que si necesitamos algo podemos recurrir siempre al otro. Gracias por todo Jesús.

Gracias a mis tías Carmen Rosa y Nieves, gracias a mi tío Carlos, mi tío preferido, y gracias a todas mis primas. Siempre habéis estado ahí para demostrarme que los lazos familiares son irrompibles. Para ayudarnos entre nosotros y para disfrutar los buenos momentos juntos. Para siempre que se necesite, estar ahí. Gracias a todos.

Gracias Carmen. Por tu sonrisa desde el primer día. Por tu apoyo, por tu saber escuchar y comprender. Por el motivarme para conseguir siempre mis objetivos. Por ser tú y por ser quien eres para mí. Por creer en mí. Por hacerme feliz. Gracias por todo.

Gracias a mi madre, Luisa, por luchar por mí en cada momento. Gracias por saber exigirme pero también saber apoyarme y enseñarme cómo superarme. Gracias por creer en mí y por haberme enseñado cómo ser una buena persona. Gracias por saber enseñarme cómo levantarse de las situaciones más complicadas. Gracias, porque soy quien soy gracias a ti, y estoy orgulloso de ello, de haber salido a ti. Gracias por tu amor. Gracias por todo.

Y gracias a mi hermano, Víctor, por tu forma de ser, por cómo nos muestras cariño siempre y por crecer a mi lado. Porque eres un hermano y también el mejor amigo. Gracias.

Jorge Yanes Marcelino

Resumen y Abstract

Resumen

En el presente Trabajo de Fin de Grado abarcaremos desde un principio el control de un brazo robótico y su modelo 3D en un entorno virtual. El objetivo será el de mover desde un script escrito en Python ambas estructuras, tanto real y virtual, al mismo tiempo, enviando desde Python el ángulo al que queremos llevar cada articulación y la articulación a mover. Estos datos serán recibidos por un controlador implementado en una placa Arduino, que moverá la estructura real, y por V-REP, programa de simulación utilizado para recrear el modelo en 3D.

Posteriormente, se intentará el control de una manera diferente. Será desde V-REP, mediante el uso de botones deslizantes, desde donde moveremos cada articulación del modelo en 3D. En Python, actuando como puente entre V-REP y Arduino, se recibirá la posición de la articulación y se enviará a Arduino. Finalmente, en Arduino se utilizará esta información para mover la estructura real según indique la posición.

En definitiva, el proyecto se basará en la comunicación de diferentes módulos softwares y el movimiento de un brazo robótico con su modelo 3D virtual al mismo tiempo.

Abstract

In this final of degree Project we will cover since the beginning the control of a robotic arm and its 3D model in a virtual environment. The objective will be to move, from a script written in Python, both structures, the real and the virtual, at same time, sending from Python the angle that we want to move every articulation and the articulation to move. These dates received by a controller implemented in an Arduino, that will move the real structure, and by V-REP, simulation program used to recreate the 3D model.

Later, it will attempt the control with a different way. It will be from V-REP, by the use of sliders, from where we will move every articulation of the 3D model. In Python, acting like a bridge between V-REP and Arduino, will receive the position of the articulation and it will be sent to Arduino. Finally, in Arduino the information will be used to move the real structure depend of the indication of the position.

Definitely, the project will be based in the communication of different modules software and the movement of a robotic arm with its 3D virtual model at the same time.

Índice

1. Introducción a los manipuladores antropomórficos.....…….…. 1

1.1. Historia de la Robótica……... 2

1.2. Partes de un manipulador antropomórfico ……. 4

1.2.1. Estructura mecánica…….... 5

1.2.2. Sistemas de actuación……..… 7

1.2.3. Sistema sensorial…….. 7

1.2.4. Controladores…….....…. 8

1.2.5. Ordenador y comunicaciones……... 8

1.2.6. Software……... 8

1.3. Manipuladores industriales según el tipo de control……... 8

1.3.1. Manipuladores…….. 9

1.3.2. Robots de repetición o aprendizaje…….. 9

1.3.3. Robots con controlador por ordenador……..…… 9

1.3.4. Robots inteligentes…… 10

1.3.5. Micro-robots……..……10

2. Estudio previo y evolución del proyecto…… 11

2.1. Evolución de la estructura física del proyecto…….. 12

2.1.1. Brazo y pinza…… 12

2.1.2. Servomotores…… 14

2.1.3. Arduino Uno…….…… 14

2.1.4. Fuente de alimentación……..15

2.1.5. Conexionado de las diferentes partes Hardware……... 17

2.2. Módulos softwares utilizados……..18

2.2.1. Arduino……..19

2.2.2. Python……19

2.2.3. V-REP……... 20

2.2.4. SketchUp…….. 21

2.3. Estabilidad y realimentación……24

3. Objetivos logrados……. 26
3.1. Movimiento de la estructura mediante Python y Arduino……. 26
3.2. Movimiento de la estructura y de un modelo con piezas puras en V-REP…… 27
3.3. Movimiento de la estructura y su representación en V-REP……. 28
3.4. Movimiento del modelo real y su representación con *sliders* desde V-REP...….. 29

4. Programación……31
4.1. Movimiento de la estructura mediante Python y Arduino……. 32
4.2. Movimiento de la estructura y de un modelo con piezas puras en V-REP…… 35
4.3. Movimiento de la estructura y su representación en V-REP……. 35
4.4. Movimiento del modelo real y su representación con *sliders* desde V-REP...….. 39

5. Simulación en V-REP…….. 42
5.1. API remota de V-REP con Python…… 47

6. Experimentos y pruebas del sistema……50
6.1. Experimentos y pruebas en la estructura física……....50
6.2. Experimentos y pruebas en el código Arduino……52
6.3. Experimentos y pruebas en el código Python……. 54
6.4. Experimentos y pruebas en V-REP……. 55
6.5. Experimentos y pruebas con potenciómetros para control de posición…….. 57

7. Presupuesto…… 59

8. Conclusions…… 60

9. Bibliografía……62

Anexo I. Códigos implementados……64

Índice de definiciones

Definición 1. Definición de robot industrial según RIA…… 1
Definición 2. Definición de robot industrial según ISO…… 1
Definición 3. Definición de robot industrial de manipulación según IFR…… 2

Índice de tablas

Tabla 1. Partes de un manipulador industrial……..2
Tabla 2. Presupuesto del presente Trabajo Fin de Grado……59

Índice de figuras

Figura 1. Modelo del robot de Leonardo Da Vinci con funcionamiento interno……3
Figura 2. Digesting Duck, Jacques de Vaucanson…….. 3
Figura 3. Robot PUMA 550/560 de Unimation con equipo de control VAL…… 4
Figura 4. Robot Humanoide ASIMO, Honda…….. 4
Figura 5. Ejemplo de estructura de un manipulador industrial. Puma 560 de Unimation.. 5
Figura 6. Tipos de articulaciones de un manipulador antropomórfico……6
Figura 7. Configuraciones más frecuentes en manipuladores industriales……. 6
Figura 8. Manipulador antropomórfico de 5GDL……. 12
Figura 9. Articulación rotacional utilizada en el brazo del presente proyecto…….. 12
Figura 10. Diagrama del brazo 1…… 13
Figura 11. Diagrama del brazo 2…… 13
Figura 12. Servomotor MG996R…… 14
Figura 13. Arduino Uno…….. 14
Figura 14. Ciclos de trabajo de una PWM en un servomotor…….. 15
Figura 15. Fuente de Alimentación ATX Power Suply 500 W…….. 16
Figura 16. Puente para arrancar una fuente de alimentación ATX Power Suply 500 W… 16
Figura 17. Conectores de una fuente de alimentación ATX Power Suply 500 W……... 16
Figura 18. Conectores y pines de una fuente de alimentación ATX Power Suply 500 W. 17
Figura 19. Conexionado de las diferentes partes Hardware…… 18
Figura 20. Arduino…….. 19
Figura 21. Python…… 19
Figura 22. V-REP…… 20
Figura 23. Modelo en V-REP con piezas puras…….. 20
Figura 24. SketchUp……... 21
Figura 25. Base creada en SketchUp…….. 22
Figura 26. Porta servo creado en SketchUp…… 22
Figura 27. Servo creado en SketchUp visualizado en V-REP…… 22
Figura 28. Eslabón de brazo creado en SketchUp…….. 23

Figura 29. Pinza creada en SketchUp visualizada en V-REP…… 23
Figura 30. Modelo en 3D recreado en SketchUp y simulado en V-REP…… 23
Figura 31. Configuración típica de control en lazo cerrado…… 24
Figura 32. Conexionado Python/Arduino……26
Figura 33. Modelo en V-REP con piezas puras…….. 27
Figura 34. Conexionado Python/V-REP con piezas puras/Arduino……... 28
Figura 35. Modelo en 3D recreado en SketchUp y simulado en V-REP…… 28
Figura 36. Conexionado Python/V-REP/SketchUp/Arduino……...…….. 29
Figura 37. Uso de *sliders* en V-REP…….. 30
Figura 38. Conexionado V-REP/SketchUp/Python/Arduino…… 30
Figura 39. *Sliders* en V-REP…….. 41
Figura 40. Pieza pura en V-REP…… 42
Figura 41. Añadir una pieza pura en V-REP…….. 42
Figura 42. Visualización de una pieza pura en V-REP…….. 43
Figura 43. Pieza importada en V-REP…… 43
Figura 44. Importación de una pieza en V-REP…… 43
Figura 45. Visualización de una pieza ".stl" importada en V-REP…… 44
Figura 46. Ventana "Object/Item position/orientation"…….. 44
Figura 47. Establecer una jerarquía en V-REP……45
Figura 48. Visualización de una jerarquía en V-REP……..45
Figura 49. Modelo creado a partir de piezas puras en V-REP…… 46
Figura 50. Modelo creado a partir de piezas ".stl" importadas en V-REP…… 46
Figura 51. Estructura del brazo con 6 gdl……... 50
Figura 52. Manipulador antropomórfico de 5GDL……51
Figura 53. Variación de la posición final debido a la inercia del movimiento del brazo....52
Figura 54. Variación tipo vaivén o resorte debido a la inercia del movimiento del brazo..53
Figura 55. Panel de usuario en Python…… 55
Figura 56. Posiciones iniciales de las articulaciones en el modelo físico……... 55
Figura 57. "Joint3" inicializada a 90º en V-REP…… 56
Figura 58. "Joint3" inicializada erróneamente por defecto a 0º en V-REP…… 56
Figura 59. Código en Arduino para comprobación de rango mediante potenciómetro….. 57
Figura 60. Conexionado Arduino con potenciómetro y servomotor……...58

1

1. Introducción a los manipuladores antropomórficos

Los robots industriales son manipuladores multifuncionales reprogramables que, hoy en día, forman parte de la mayoría de procesos industriales, facilitando la realización de diversas tareas. Existen diferentes definiciones para el término de robot industrial o manipulador. La definición comúnmente más aceptada viene dada por la Asociación de Industrias Robóticas (RIA) como:

"Un robot industrial es cualquier estructura mecánica que opera con un cierto grado de autonomía, bajo el control de un computador, para la realización de una tarea, y que dispone de un sistema sensorial evolucionado para obtener información del entorno".

Definición 1. Definición de robot industrial según RIA [1], [12]

Por otro lado, la Organización Internacional de Estándares (ISO), modificando ligeramente la definición dada por la RIA, incluye en esta definición la necesidad de que el robot tenga varios grados de libertad. De este modo determina al robot industrial como:

"El robot industrial es un manipulador multifuncional reprogramable con varios grados de libertad, capaz de manipular materias, piezas, herramientas o dispositivos especiales, según trayectorias variables programadas para realizar tareas diversas".

Definición 2. Definición de robot industrial según ISO [7]

Por último, la Federación Internacional de Robótica (IFR) hace distinción entre un robot industrial de manipulación y otro tipo de robots. En esta definición además se debe entender que la reprogramabilidad y la multifunción se consiguen físicamente por el robot:

"Se entiende como robot industrial de manipulación una máquina de manipulación automática, reprogramable y multifuncional con tres o más ejes que pueden posicionar y orientar materias, piezas, herramientas o dispositivos especiales para la ejecución de trabajos diversos en las diferentes etapas de la producción industrial, ya sea en una posición fija o en movimiento".

Definición 3. Definición de robot industrial de manipulación según IFR [6]

Hay muchas otras definiciones según cada institución. En resumen, podríamos entender la definición de un robot industrial como la de un manipulador multifuncional reprogramable compuesto por una estructura mecánica que, operando con un cierto grado de autonomía bajo el control de un computador, es capaz de realizar una tarea como por ejemplo mover materias, piezas, herramientas o dispositivos especiales, según trayectorias variables programadas. El manipulador dispondrá, a su vez, de un sistema sensorial que le habilite para obtener información de su entorno. Por lo tanto, se podría decir que un manipulador, en su conjunto, está formado por software y hardware.

Hardware	Software
Estructura mecánica	Gestión del sistema sensorial.
Sistema de actuación	
Sistema sensorial	Movimiento de la estructura mecánica para la realización de una tarea
Controladores y Ordenador	

Tabla 1. Partes de un manipulador industrial [1]

1.1. Historia de la Robótica

La robótica, en muchos de sus aspectos, está orientada a la creación de máquinas que guardan semejanza o que pueden realizar las mismas tareas que los seres humanos. Es con Karel Capek, en 1921, donde se acuña el término "robot" a unos seres que realizaban tareas rutinarias y desagradables para un ser humano en su obra Rossun Universal Robot [1], [19].

A continuación nombraremos una serie de fechas importantes dentro de la historia de la robótica.

- 1495: Leonardo Da Vinci diseña el "Caballero mecánico", un robot humanoide.

Figura 1. Modelo del robot de Leonardo Da Vinci con funcionamiento interno [1], [18]

- 1738: Jacques de Vaucanson y su Digesting Duck, pato mecánico capaz de moverse.

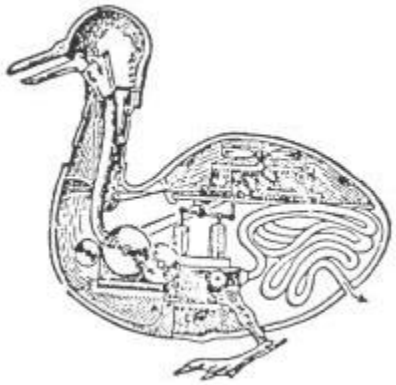

Figura 2. Digesting Duck, Jacques de Vaucanson [1], [19]

- 1921: Término Robot, Karel Capek en su obra Rossun Universal Robot.
- 1954: Primer robot comercial de la compañía Unimation fundada por George Devol. En principio realizaban tareas de manipulación donde el manipulador se encontraba fijo e imitaba al brazo humano.

Figura 3. Robot PUMA 550/560 de Unimation con equipo de control VAL [1]

- 2000: Robot Humanoide ASIMO fabricado por Honda.

Figura 4. Robot Humanoide ASIMO, Honda [5], [17]

1.2. Partes de un manipulador antropomórfico

Según la finalidad a la que está orientado el manipulador, las partes, tanto hardware como software, variarán para facilitar la realización de la tarea indicada de la manera más cómoda posible. Por ejemplo, si se trata de un proceso industrial con control por parte de un operario es mejor un software o hardware de control directo que contenga el menor número de comandos, palancas, botones… posibles para facilitar su gestión. Por otro lado, si nos vamos a un robot de autoaprendizaje el software será programado internamente dentro de los controladores del robot, será mucho más complejo y necesitará de un programador cualificado.

1.2.1. Estructura mecánica

La estructura mecánica de un manipulador está formada por las piezas que componen cada uno de los eslabones del robot y los tipos de articulaciones que unen estas piezas. Su constitución se asemeja, en la mayoría de los robots industriales, con las extremidades superiores del cuerpo humano. Por esto, se suelen utilizar los términos como cintura, hombro, brazo, codo, muñeca, mano… para nombrar a las diferentes partes de un manipulador industrial.

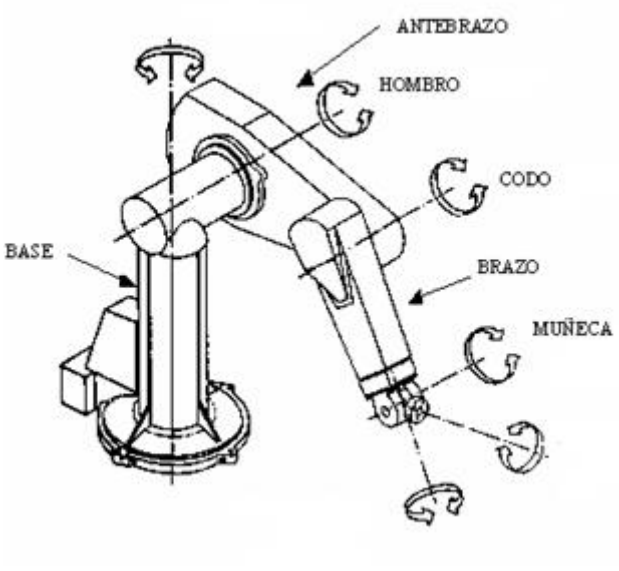

Figura 5. Ejemplo de estructura de un manipulador industrial. Puma 560 de Unimation [1]

Las piezas o eslabones normalmente tienen formas geométricas como cilíndricas y cúbicas para facilitar su control y la gestión del espacio que ocupan así como su montaje, pero pueden variar según las necesidades.

Por otro lado, las articulaciones son las que se ocupan de clasificar a los manipuladores industriales. Dependiendo de cada tipo de articulación se tienen una serie de grados de libertad y, gracias a éstos, se realizan diferentes tareas según el tipo de movimiento que se necesite. Las más comunes se clasifican en:

- Rotación (1 grado de libertad).

- Prismática (1 grado de libertad).
- Cilíndrica (2 grados de libertad).
- Planar (2 grados de libertad).
- Esférica o rótula (3 grados de libertad).

A continuación, veremos una imagen donde se detalla cada una de los tipos de articulaciones posibles.

Figura 6. Tipos de articulaciones de un manipulador antropomórfico [8]

De este modo, las diferentes combinaciones de los tipos de articulaciones implican distintas configuraciones del brazo en manipuladores industriales. Las más comunes son:

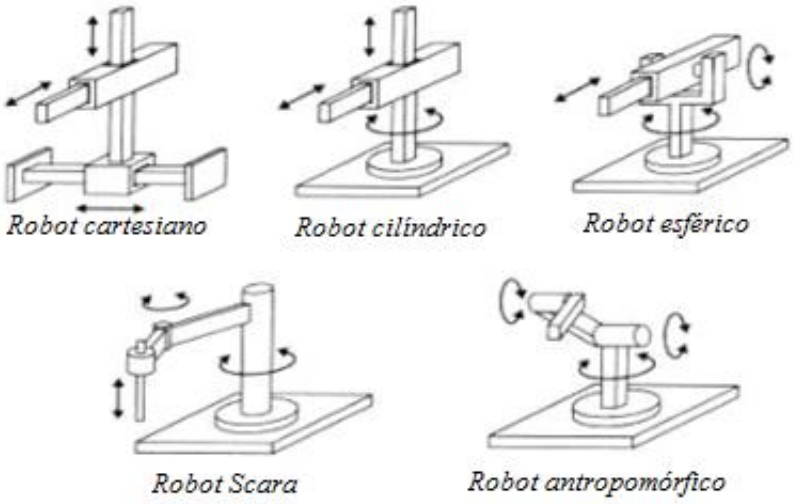

Figura 7. Configuraciones más frecuentes en manipuladores industriales

Cada articulación provee al manipulador, de al menos, un grado de libertad. Los grados de libertad son cada uno de los movimientos independientes (giros y desplazamientos) que puede realizar cada articulación con respecto a la anterior y son parámetros que se precisan para determinar la posición y orientación del efector final del manipulador. Además, el número de grados de libertad de un manipulador coincide con la suma de todos los grados de libertad de cada articulación del robot. En contra, cuando el número de grados de libertad es mayor que los necesarios para realizar una determinada tarea se dice que el robot es redundante.

1.2.2. Sistema de actuación

Los sistemas de actuación se encargan de generar las fuerzas o pares que necesita el brazo para moverse. Pueden ser sistemas hidráulicos, neumáticos y motores eléctricos, en particular los motores de corriente continua servocontrolados. A su vez, hay sistemas de transmisión que llevan el movimiento a las distintas piezas como son, por ejemplo, engranajes y correas.

1.2.3. Sistema sensorial

El sistema sensorial está ligado directamente al sistema de control, cual responderá mejor a más información tenga del entorno. Esta información puede ser adquirida por medio de sensores, ya sean infrarrojos, de sonido e incluso cámaras que detectan objetos, que envían los datos de estos al brazo. Por otro lado, debe existir un sistema de realimentación en lazo cerrado para comprobar la posición del manipulador en cada momento y así garantizar el mínimo error en su posición actual.

La autonomía de un robot se basa en un sistema de navegación autónomo. Un robot de este tipo deberá tener el mayor número de sensores posibles para adquirir del entorno la mayor cantidad de información posible. Dado esto, en un manipulador podemos destacar problemas como la tarea a realizar, la trayectoria a seguir y evitar los obstáculos que se encuentre por la misma trayectoria. A partir de los sistemas sensoriales, el sistema de control facilitará la realización de la tarea mediante una programación del movimiento adecuada.

1.2.4. Controladores

Los controladores son los encargados de recoger la información que reciben del sistema sensorial y enviarle órdenes al sistema de actuación encargándose de conectar la estructura del manipulador con un ordenador o panel que es controlado por el usuario. Normalmente están compuestos de un circuito impreso unido a un microcontrolador y a varios puertos de entrada/salida. Existen diferentes modelos de controladores y empresas que se encargan del sistema de control de los manipuladores antropomórficos.

1.2.5. Ordenador y comunicaciones

En el ordenador es donde encontramos los programas o interfaces para actuar sobre el controlador, y que el usuario u operario pueda enviar órdenes y/o comandos al brazo. A su vez, se muestra información recibida por el sistema sensorial para poder actuar en base a cada situación de una manera distinta. Por otro lado, las comunicaciones pueden ser por cable o inalámbricas, analógicas o digitales, etc. Cada una de las posibles maneras de comunicación se aplicará según los requerimientos de cada tarea.

1.2.6. Software

Los softwares utilizados para los controles de los manipuladores antropomórficos son diversos. Podemos encontrarnos programaciones para que el robot sea autónomo, simuladores visuales, programas con controles mediante “sliders” o botones digitales, programas que muestran la información captada por el sistema sensorial del robot… Cada software será utilizado según la finalidad del manipulador.

1.3. Manipuladores industriales según el tipo de control

Existen varios tipos de robots o manipuladores industriales que se clasifican según el objetivo para el que han sido fabricados. Se podrían englobar en:

- Manipuladores.
- Robots de repetición y aprendizaje.
- Robots con control por ordenador.
- Robots “inteligentes”.
- Micro-robots.

1.3.1. Manipuladores

Los manipuladores constan de una estructura mecánica con un sistema de control, normalmente sencillo, para facilitar el control del movimiento de cada uno de sus elementos. Tiene varios modos de control:

- Manual: El operario controla directamente la tarea del manipulador.
- De secuencia fija: Modo en el que se repite la trayectoria o proceso de trabajo preparado previamente.
- De secuencia variable: Modo en el que se pueden alterar algunas características de los ciclos de trabajo así como parte de su trayectoria o tarea.

1.3.2. Robots de repetición o aprendizaje

Este tipo de robots son manipuladores que repiten una serie de movimientos previamente ejecutada por un operador humano haciendo uso de un controlador manual. Se entienden entonces dos fases, una de aprendizaje donde el operario enseña al robot (moviéndolo con dispositivos, usando algún guante o maniquí, o incluso moviendo el brazo directamente), y una fase de repetición donde el robot repite el movimiento anteriormente programado. Son robots muy utilizados en diferentes sistemas industriales y su tipo de programación recibe el nombre de “gestual”.

1.3.3. Robots con controlador por ordenador

Manipuladores mecánicos controlados por un ordenador y con un microcontrolador como etapa media para administrar los datos recibidos del entorno y enviar las órdenes a los actuadores. Este tipo de manipuladores tiene un programa en un lenguaje específico que recoge las diferentes instrucciones adaptadas para el robot. Esta programación es denominada "textual" y es creada sin la intervención del manipulador. Este tipo de robot exige una programación por parte de personal cualificado.

1.3.4. Robots inteligentes

Son similares a los robots con controlador pero éstos son capaces de obtener información del entorno y tomar decisiones en tiempo real, autorreprogramándose.

1.3.5. Micro-robots

Robots con fines educacionales, de entretenimiento, o de investigación que se distinguen por un precio más económico y cuya estructura y funcionamiento son similares a los de aplicación industrial.

2

2. Estudio previo y evolución del proyecto

Los manipuladores antropomórficos son altamente utilizados en la industria dada su capacidad de realizar tareas repetitivas con una gran velocidad y una producción constante, facilitando el trabajo al operario.

2.1. Evolución de la estructura física del proyecto

Como punto de partida, se tenía el desarrollo de un prototipo de manipulador antropomórfico, compuesto por brazo, antebrazo y mano creados mediante una impresión en 3D. La mano constaría de dedos cual tendría un movimiento tipo "manopla".

Debida las dificultades tanto temporales como económicas que suponía la impresión en 3D del brazo y la mano, se planteó, tras su estudio, realizar un modelo virtual del brazo con una pinza, dejando de lado la estructura física.

En cambio, posteriormente, gracias a una estructura de bajo coste, se resolvió por la utilización de un modelo de piezas rígidas ya fabricado con una pinza en vez de mano como efector final. De este modo se conseguiría reproducir los movimientos del modelo virtual en el modelo físico. A su vez, para mover dicha estructura se utilizarían servomotores como actuadores de cada una de las articulaciones del brazo y la pinza.

2.1.1. Brazo y pinza

La estructura final sería un brazo antropomórfico con pinza, realizada con un metal ligero pero rígido que facilita su movimiento, visualizada en la figura 10:

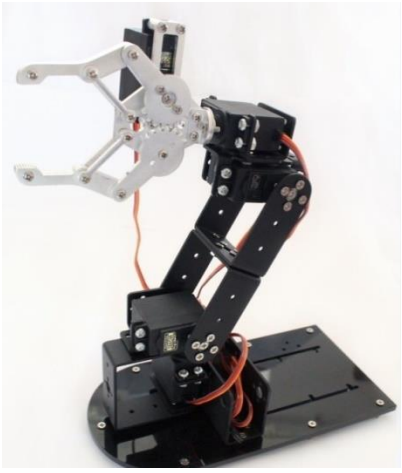

Figura 8. Manipulador antropomórfico de 5GDL

El manipulador antropomórfico del presente Trabajo Fin de Grado tiene la posibilidad de tener hasta 6 grados de libertad, con posibilidad de variación. Se ha optado por elaborar la estructura con 5 grados de libertad. De esta manera no se realiza un manipulador antropomórfico redundante y no se lleva a los servomotores utilizados a su límite de carga.

Las cinco articulaciones utilizadas son articulaciones rotacionales, tres están situadas en el brazo y dos en la pinza (una para la rotación localizada en la base de la pinza y otra para la apertura y cierre de la pinza mediante un sistema de engranajes que conecta las dos partes de la pinza).

Figura 9. Articulación rotacional utilizada en el brazo del presente proyecto [8]

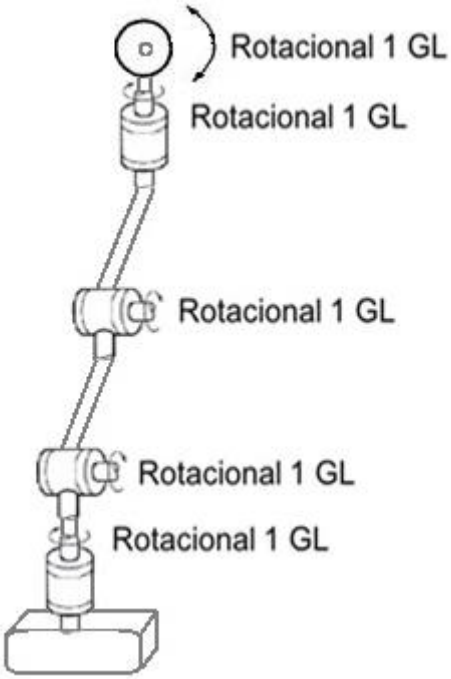

Figura 10. Diagrama del brazo 1

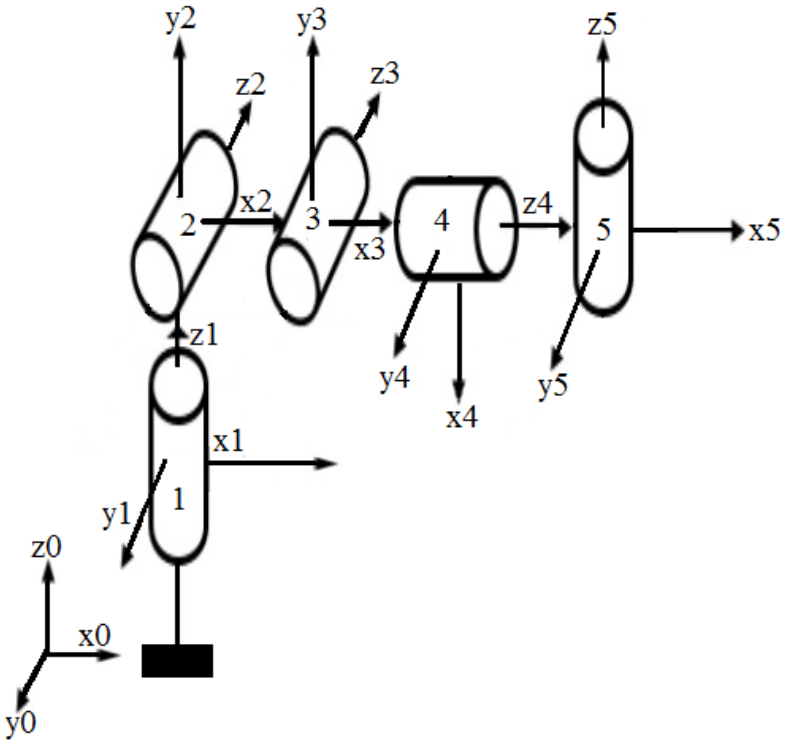

Figura 11. Diagrama del brazo 2

2.1.2. Servomotores

Con la estructura mecánica a utilizar se añadirán una serie de servomotores. Éstos serán servomotores MG996R, que tienen un torque de 9,4 kg-cm a 4,8 y una velocidad máxima de 0,19 segundos/60º a 4,8 V. Se ha elegido estos servomotores porque se adaptaban bien a la estructura del brazo gracias a su pequeño tamaño y porque tenían el torque necesario para poder realizar los movimientos de cada articulación del manipulador sin ningún problema de carga.

Figura 12. Servomotor MG996R

2.1.3. Arduino Uno

Arduino es una compañía de hardware y software libre que diseña y manufactura piezas de desarrollo de hardware y software, compuesta por circuitos impresos que integran un microcontrolador y un entorno de desarrollo (IDE) donde se programa cada placa [2]. Toda la plataforma Arduino es de licencia de código abierto.

El hardware consiste en un circuito impreso con microcontrolador (en la mayoría de los casos un Atmel AVR) con una serie de puertos digitales y analógicos de entrada/salida.

Figura 13. Arduino Uno [2]

El microcontrolador utilizado será Arduino Uno, conectado por serial al ordenador y utilizando sus pines que permiten PWM, que son los que utilizaremos para llevar al servomotor al ángulo deseado.

La PWM o modulación por ancho de pulsos de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (por ejemplo senoidal o cuadrada) para transmitir información o controlar la cantidad de energía que se envía a una carga. La explicación gráfica para una señal PWM en los ciclos de trabajo de un servomotor sería:

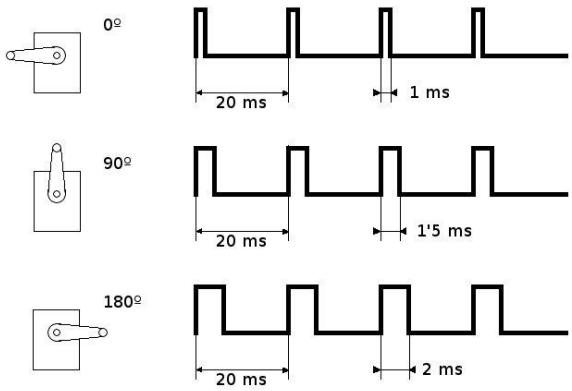

Figura 14. Ciclos de trabajo de una PWM en un servomotor

2.1.4. Fuente de alimentación

Se utilizará una fuente de alimentación externa para poder generar los voltajes y las intensidades de corriente necesarias para que cada uno de los servomotores sea capaz de moverse. Hay que tener en cuenta que la tierra "GND" de la Arduino tiene que estar conectada a la tierra de la fuente de alimentación para un correcto funcionamiento de los servomotores.

En este caso, para optar por una solución económica se ha procedido a elegir una fuente de alimentación de un ordenador común de sobremesa, de un bajo coste en comparación con

otras fuentes de alimentación externas, pero capaz de proporcionar la corriente necesaria. El modelo es una ATX Power Suply 500 W.

Figura 15. Fuente de Alimentación ATX Power Suply 500 w.

Por otro lado, las fuentes de alimentación necesitan un puente entre el conector "Power On" (cable verde) y una de las tierras del mismo conector "COM" (cable negro).

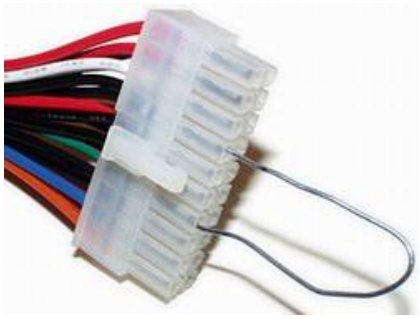

Figura 16. Puente para arrancar una fuente de alimentación ATX Power Suply 500 W

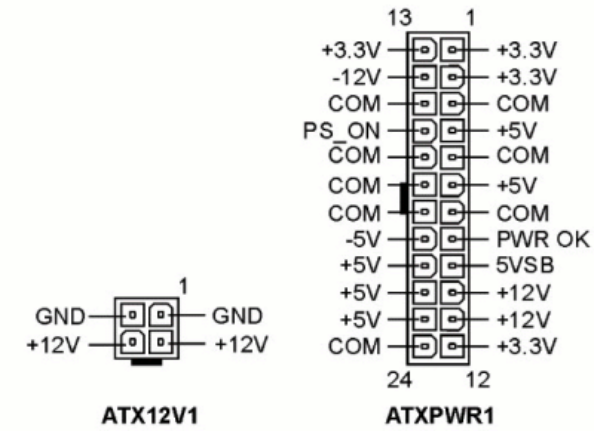

Figura 17. Conectores de una fuente de alimentación ATX Power Suply 500 W

24-pin ATX power supply connector
(20-pin omits the last 4: 11, 12, 23 and 24)

Color	Signal	Pin	Pin	Signal	Color
	+3.3 V	1	13	+3.3 V sense	
	+3.3 V	2	14	-12 V	
	Ground	3	15	Ground	
	+5 V	4	16	Power on	
	Ground	5	17	Ground	
	+5 V	6	18	Ground	
	Ground	7	19	Ground	
	Power good	8	20	-5 V	
	+5 V standby	9	21	+5 V	
	+12 V	10	22	+5 V	
	+12 V	11	23	+5 V	
	+3.3 V	12	24	Ground	

Figura 18. Conectores y pines de una fuente de alimentación ATX Power Suply 500 W

El conector PS_ON lo usa la placa base un ordenador para indicarle a la fuente de alimentación que se encienda. Su activación es mediante la bajada de su voltaje a 0 V. Por esto, al puentearlo con un cable de toma de tierra la fuente arranca.

2.1.5. Conexionado de las diferentes partes Hardware

Se conectarán los servomotores del brazo a la alimentación y los diferentes pines PWM de la placa Arduino. A su vez la GND de la Arduino también se conecta a la GND de la fuente de alimentación.

Por otro lado, la placa Arduino se conecta al puerto USB/serial del ordenador, para facilitar el envío de datos a cada uno de los servomotores.

A continuación se muestra un esquemático de la conexión de la placa Arduino con la fuente de alimentación, los servomotores y el ordenador.

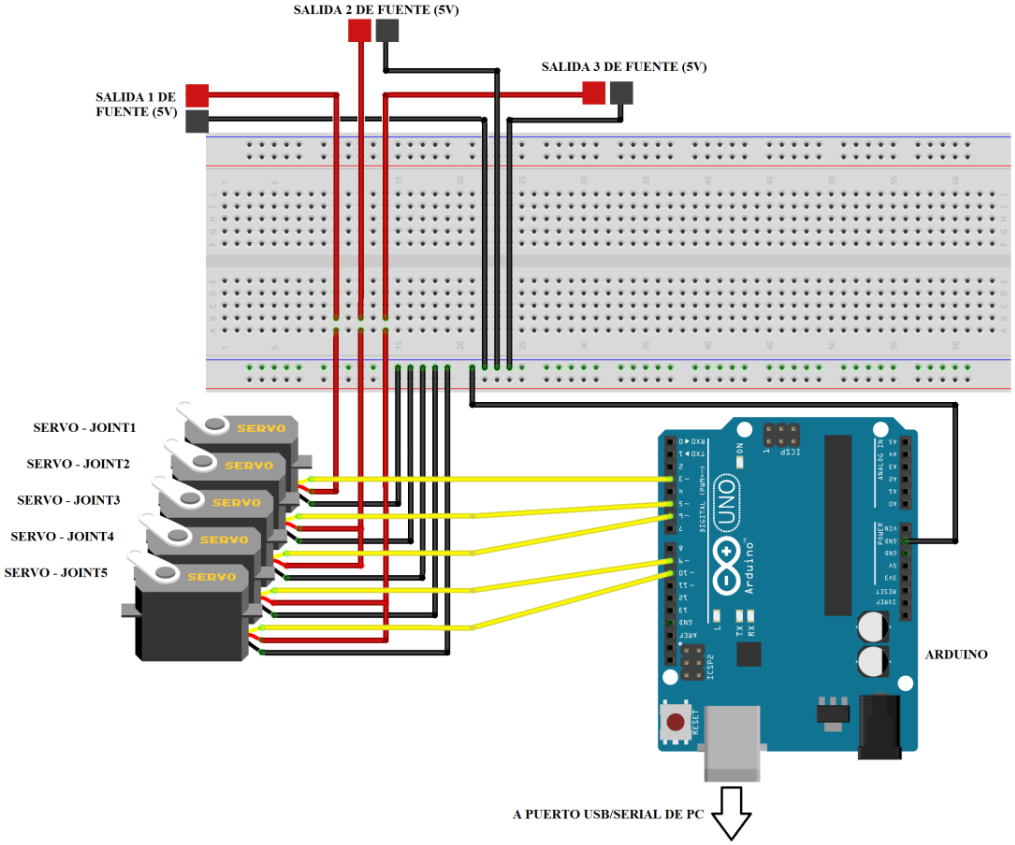

Figura 19. Conexionado de las diferentes partes Hardware

2.2. Módulos softwares utilizados

En un principio se realizaría un control de bajo nivel basado en plataformas Arduino para los actuadores del brazo y un control de alto nivel desde un ordenador en un entorno como MATLAB. Dada la evolución del proyecto, se cambiaría el entorno MATLAB por una programación en PYTHON y se añadiría la utilización del modelo recreado en 3D mediante el uso de SketchUp (piezas exportadas en formato ".stl") en un programa de simulación como V-REP.

Por lo tanto, los módulos softwares utilizados a lo largo del proyecto serán Arduino, Python, V-REP y SketchUp.

2.2.1. Arduino

El software de Arduino Genuino es utilizado para realizar el conexionado de cada uno de los puertos habilitados para una señal PWM con cada servomotor, y controlar el giro de los servomotores. Desde este código se actúa directamente sobre Arduino mediante el puerto serial al que está conectado el microcontrolador y enviándose las posiciones a cada uno de los servomotores.

En la programación en Arduino también se utilizará la librería "VarSpeedServo.h" para controlar la velocidad de giro de los servomotores. Este control de velocidad resulta útil para realizar los movimientos del brazo a un tiempo mucho más perceptible, manteniendo también en buen estado a los servomotores, cuales podrían sufrir daños si se realizan giros bruscos y con cargas de peso (la misma carga del brazo).

Figura 20. Arduino [2]

2.2.2. Python

En Python Spyder se realizará la mayor parte del código necesario para la programación del manipulador antropomórfico. Su función será recoger las diferentes posiciones o enviarlas tanto al brazo físico como al modelo creado y simulado en V-REP. Se podría decir que el código en Python actúa como un puente entre la interfaz virtual V-REP y el control directo del brazo en Arduino.

Figura 21. Python [9]

2.2.3. V-REP

En V-REP versión Edu, donde se ha elegido su versión educativa ya que es gratuita, se implementará cada una de las piezas del brazo robótico, simulando su funcionamiento real ajustando cada una de las funciones necesarias para su correcto funcionamiento. La simulación tendrá una cinemática directa controlada desde Python y una cinemática directa controlada directamente desde V-REP. Para este último caso, existirá un código implementado en LUA.

Figura 22. V-REP [13]

Además, en uno de los objetivos se implementará un brazo que simula el brazo físico. Esta simulación será creada con piezas puras de V-REP, por lo que la similitud será mucho menor

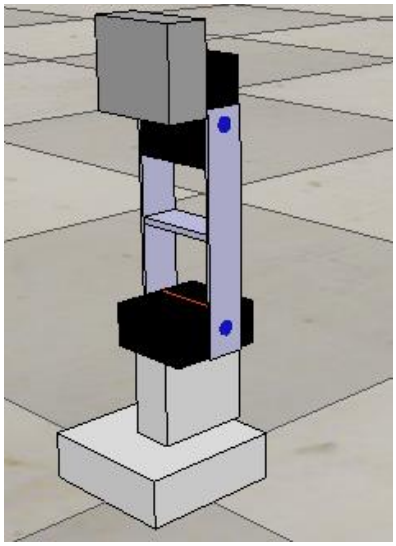

Figura 23. Modelo en V-REP con piezas puras

2.2.4. SketchUp

Cada una de las piezas será recreada en 3D para luego pasar los archivos generados en ".stl" a V-REP y formar el conjunto del brazo. Existen diversas herramientas que permiten generar modelos en 3D. Blender, SketchUp e incluso el mismo V-REP (que permite crear figuras geométricas básicas) son diferentes herramientas que permiten la edición en 3D.

Según las características de cada uno de los programas, Blender sería el que ofrecería una mayor rigurosidad y calidad del modelo pero en contra veríamos la dificultad de creación y manejo del programa, así como el peso de cada una de las figuras, que tendría una mayor repercusión en la dificultad de simular sus movimientos en el programa V-REP.

Posteriormente, V-REP permite la creación de piezas cilíndricas, cúbicas, planos… con una facilidad mucho mayor para la simulación, ya que las trata como piezas puras, cuales en este entorno son las más fáciles de simular. En contra tendríamos que el aspecto visual del modelo no sería tal cual el brazo antropomórfico real. Aun así, recrearemos el brazo antropomórfico como primero paso con piezas puras creadas en el V-REP (sin representar la pinza, cual requeriría un grado de precisión mucho mayor) para simular el movimiento del brazo en el programa V-REP y ver las similitudes entre el movimiento simulado y el movimiento real.

Finalmente, escogeríamos el programa SketchUp porque, aparte de ser un software de acceso gratuito, tendría el punto intermedio entre la calidad visual y facilidad de uso. De esta manera se consiguen crear piezas viables para la simulación y que mantengan una similitud con el modelo real.

Figura 24. SketchUp [11]

El modelo en SketchUP que se proporciona a V-REP viene creado por diferentes links como se pueden ver en las siguientes figuras 25, 26, 27, 28 y 29:

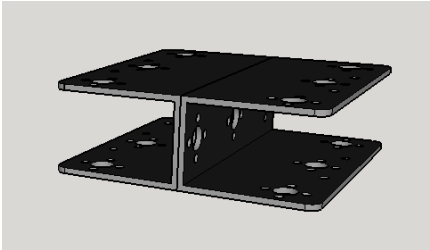

Figura 25. Base creada en SketchUp

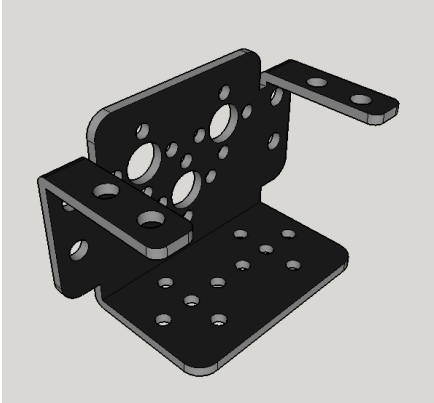

Figura 26. Porta servo creado en SketchUp

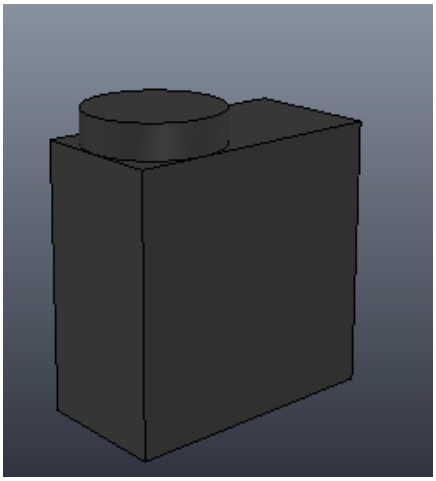

Figura 27. Servo creado en SketchUp visualizado en V-REP

Figura 28. Eslabón de brazo creado en SketchUp

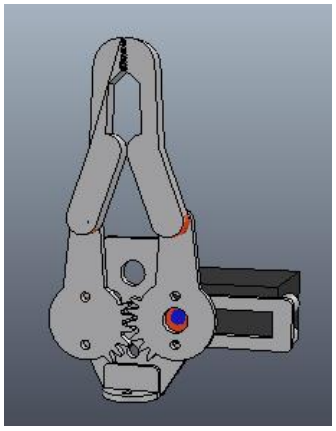

Figura 29. Pinza creada en SketchUp visualizada en V-REP (la pinza está realizada por cinco piezas)

Tras la unión de todas y cada una de las piezas creadas en SketchUp, se obtiene el modelo virtual en 3D que se asemeja a la estructura física del presente proyecto.

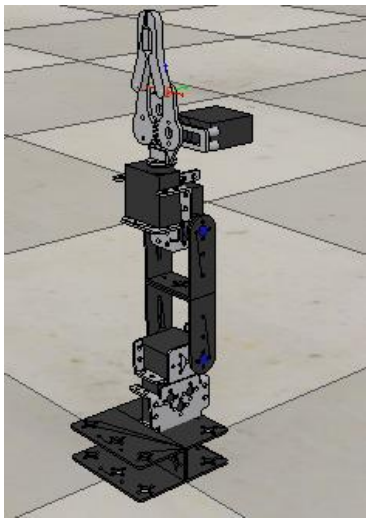

Figura 30. Modelo en 3D recreado en SketchUp y simulado en V-REP

2.3. Estabilidad y realimentación

En los manipuladores industriales existe normalmente un sistema de realimentación en lazo cerrado para conocer cada una de las posiciones y los estados de las piezas del manipulador. Esto se debe para un control lo más exacto posible de la estructura y minimizar los fallos, intentando alcanzar lo más rápido posible y de manera adecuada el valor de consigna al que el sistema desea llegar.

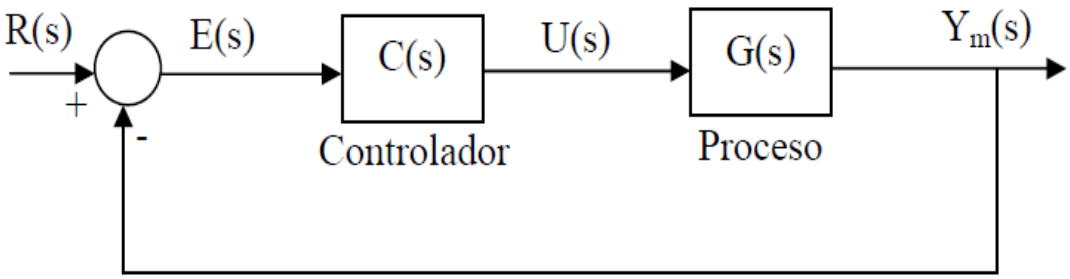

Figura 31. Configuración típica de control en lazo cerrado

Pero a un sistema de control en lazo cerrado también le afectan otros factores que pueden causar inestabilidad como son el retardo o las interferencias o ruidos. En un manipulador industrial el retardo afectará en la rapidez que la estructura alcanza la posición indicada. Por otro lado, los ruidos podrían existir varios, como el rozamiento entre piezas, un desvío en la posición correcta debido al propio peso del manipulador, la precisión y capacidad del servomotor o motor de llegar a un ángulo concreto… Además, cuando se utiliza una interfaz, tanto hardware como software, para controlar el manipulador, pueden existir otra serie de retardos (velocidades de sincronismo entre las partes) o ruidos. Dado esto, los manipuladores industriales deben considerar cada uno de estos aspectos para, de este modo, poder trabajar dentro de la estabilidad y de manera adecuada. En cambio, para este proyecto no tendremos en consideración las perturbaciones que se puedan ocasionar ya que nos centraremos en cómo responden la estructura física y el modelo en V-REP a la variación de la posición de las articulaciones.

Además, partiendo de la premisa de que los servomotores llegan a la posición indicada en todo momento, en el presente proyecto no se realiza un circuito de realimentación en lazo cerrado por medio de sensores o potenciómetros que indiquen la localización física de cada una de las posiciones, después de haberles indicado que vayan a dicha posición. Por otro lado, sí se complementa con un "lazo cerrado" visual, la representación en V-REP, que indica si la estructura física ha llegado, al menos aproximadamente, a la posición que marca el modelo en 3D.

3

3. Objetivos logrados

Se ha conseguido una serie de objetivos gracias a la diversa programación y control del manipulador antropomórfico desarrollado en el presente Trabajo Fin de Grado.

3.1. Movimiento de la estructura mediante Python y Arduino

Se ha conseguido el movimiento de la estructura mediante un código en Python que se conecta y envía a Arduino los grados a los que llevar cada una de las articulaciones. En Arduino se implementa la asignación de los pines de entrada/salida a cada servomotor de la estructura y se programa un control de velocidad mediante la librería "VarSpeedServo.h".

En la siguiente figura 32 se indica el conexionado para este proceso.

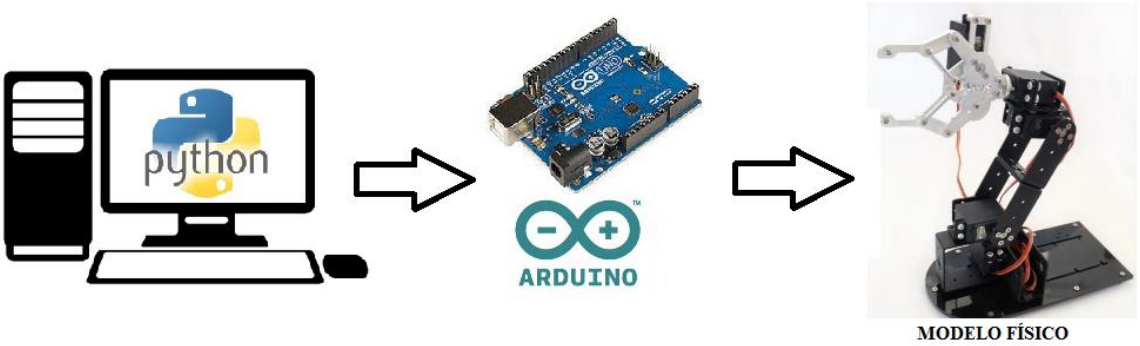

Figura 32. Conexionado Python/Arduino

3.2. Movimiento de la estructura y de un modelo en 3D con piezas puras V-REP.

En este paso se ha conseguido el movimiento de la estructura creada a partir de piezas puras de V-REP al "mismo tiempo" que se mueve el brazo físico. Desde Python se envían las posiciones de cada una de las articulaciones (cinemática directa). Estas posiciones se envían tanto al modelo de V-REP como al controlador Arduino para el movimiento del brazo.

El modelo recreado simula cada uno de los eslabones del brazo sin la pinza. Con este modelo se realizan las pruebas necesarias para visualizar la posible respuesta del movimiento de la estructura física al mismo tiempo que un modelo virtual en V-REP con el uso de sus piezas puras, que se basan en planos, discos, cubos, esferas y cilindros.

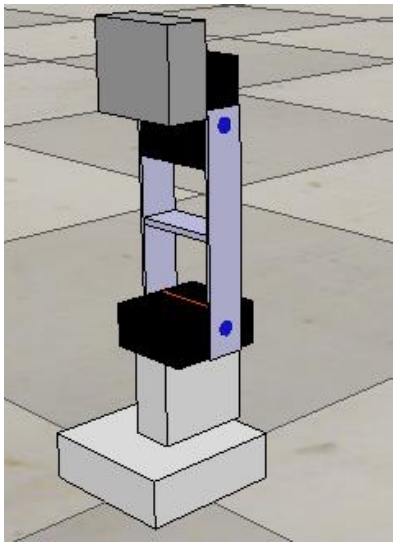

Figura 33. Modelo en V-REP con piezas puras

En este caso, el diagrama sería tal y como se muestra en la figura 34.

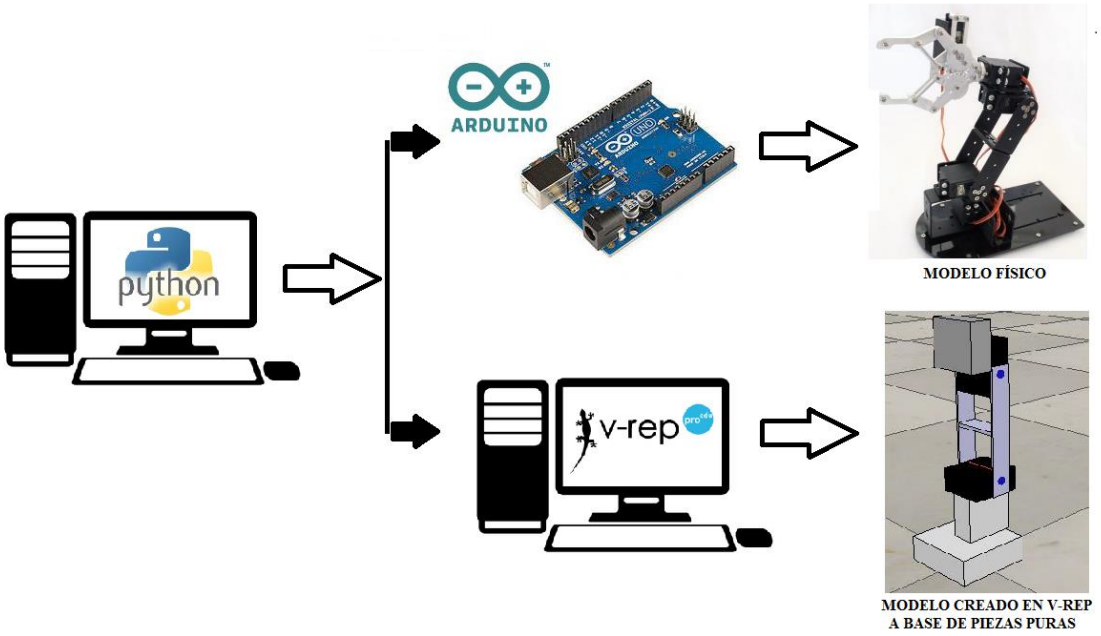

Figura 34. Conexionado Python/V-REP con piezas puras/Arduino

3.3. Movimiento de la estructura y su representación en V-REP.

En este caso, al igual que en el apartado anterior, se ha conseguido mover la estructura al mismo tiempo que se mueve su recreación en un modelado 3D en V-REP. Este modelo virtual ha sido realizado por piezas separadas en el entorno SketchUp. Se le envían desde Python, tanto a la estructura como a su representación en 3D en V-REP, las posiciones en grados de cada una de las articulaciones. Una vez seleccionada la articulación y la posición en grados a la que debe llegar, se obtiene un resultado en el modelo físico y su representación.

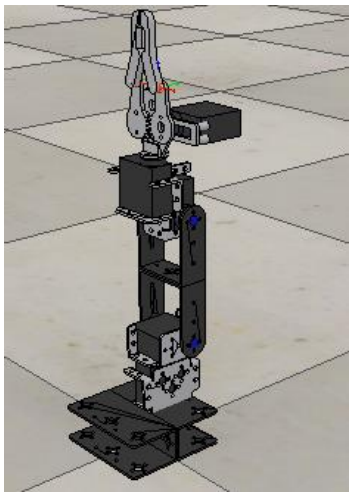

Figura 35. Modelo en 3D recreado en SketchUp y simulado en V-REP

Vemos como sería la implementación de los diferentes módulos softwares en este caso en la figura 36.

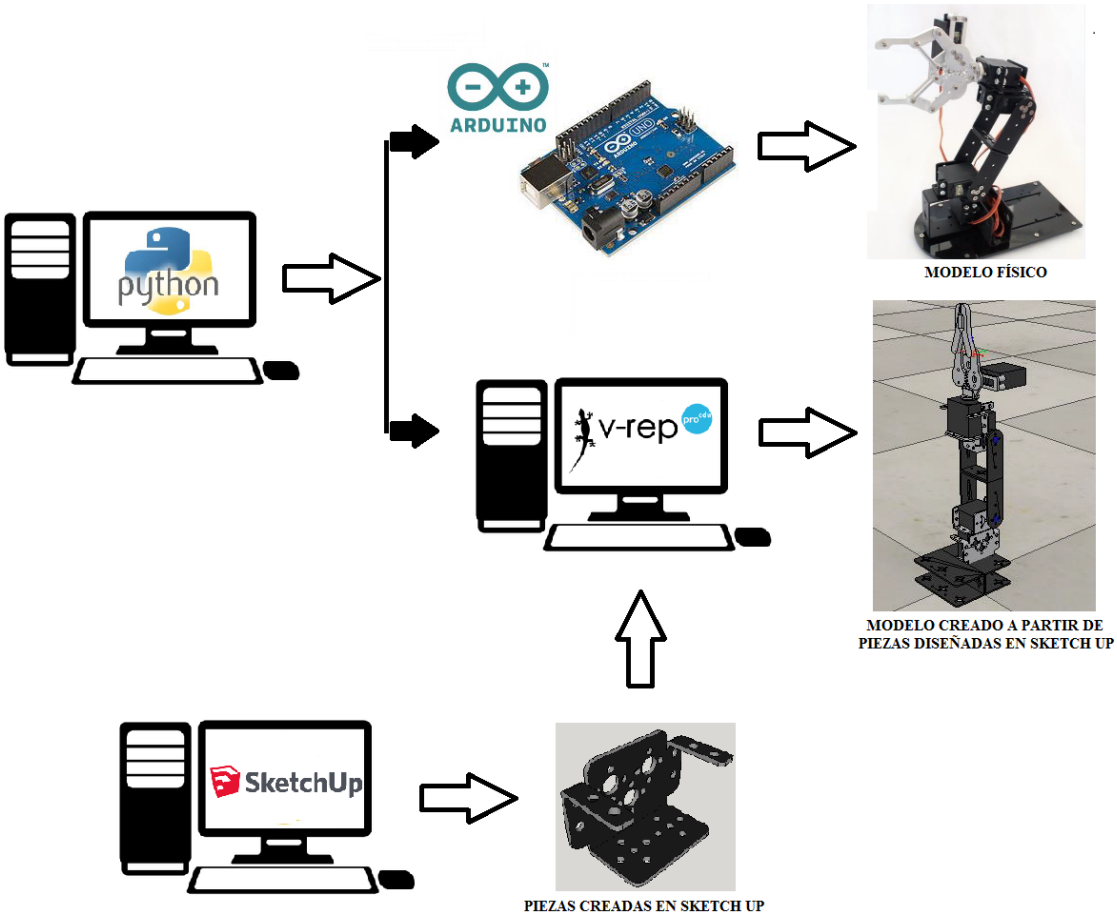

Figura 36. Conexionado Python/V-REP/ SketchUp/Arduino

3.4. Movimiento del modelo real y su representación con *sliders* desde V-REP.

Se ha conseguido un control de la estructura y de su recreación en V-REP mediante el uso de *sliders* programados en el entorno V-REP. En este caso se le envía una posición mediante los cursores deslizantes programados en V-REP. A su vez, existe un código en Python que recibe a la posición de la articulación del modelo en 3D, y obteniendo dicho dato, lo envía a la Arduino para mover la estructura física.

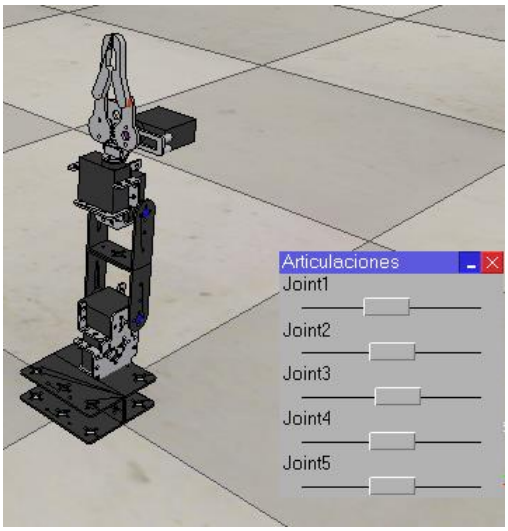

Figura 37. Uso de *sliders* en V-REP

La figura 38 nos muestra el diagrama referente en el caso de que V-REP mande sobre Python.

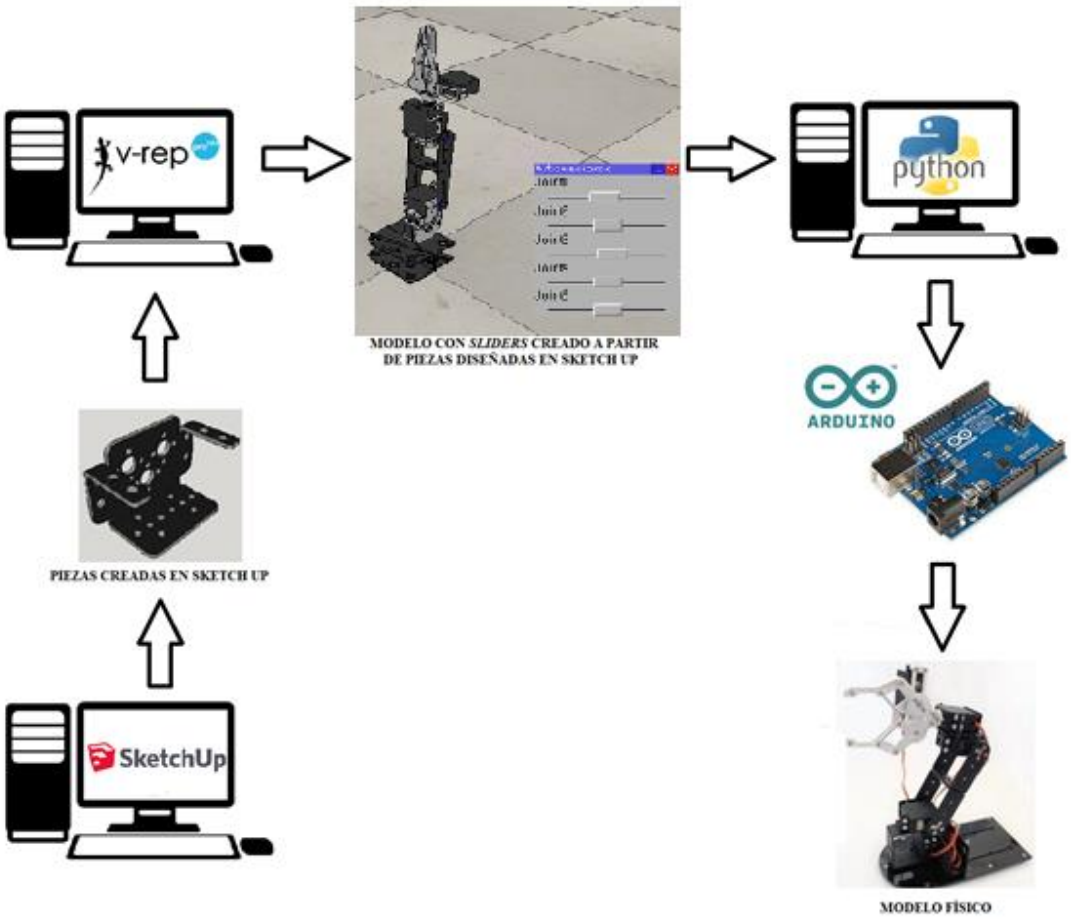

Figura 38. Conexionado V-REP/SketchUp/Python/Arduino

4

4. Programación

Para cada uno de los objetivos logrados existen diferentes códigos en Arduino, Python y V-REP (LUA). En este capítulo estudiaremos y describiremos el proceso y funcionamiento de las partes más relevantes del código en cada uno de los objetivos logrados. Si se desea visualizar el código completo comentado, recurrir al "Anexo I. Códigos implementados".

El código en Arduino es general y usado por todos y cada uno de los objetivos menos en el caso de control mediante *sliders* de V-REP. Con el código se envían los ángulos a los servomotores, cuales están conectados a los diferentes pines del controlador Arduino.

En primer lugar se implementaría la librería "VarSpeedServo.h" y se declararían cada uno de los servos con esta librería [16].

```
#include <VarSpeedServo.h>

VarSpeedServo servo1;
VarSpeedServo servo2;
VarSpeedServo servo3;
VarSpeedServo servo4;
VarSpeedServo servo5;
```

La librería utilizada se utiliza tal y como se ve en el código a continuación:

```
servo1.write(angle, 40, true);
```

De esta manera se envía la posición como un dato en una señal PWM mediante un puerto serial al que está conectada la placa Arduino.

Como vemos en el ejemplo del "servo1", en base al funcionamiento de la librería, se dan tres valores al servo mediante un "write". Estos valores son el ángulo, la velocidad y una variable booleana que en caso de ser *true* hace que la función se bloquee hasta que se realice el movimiento.

Por otro lado, la velocidad va de 1 a 255, siendo a mayor número, mayor velocidad. El valor 0 es la máxima velocidad permitida por el servomotor. Además, cada servomotor puede tener una velocidad de movimiento distinta.

También en Arduino se asignan los pines de la placa a cada servomotor de manera que:

```
servo1.attach(3, minPulse, maxPulse);
```

En el ejemplo del "servo1" se le otorga al primer servomotor el pin 3. Todos los pines utilizados permitirán PWM. A su vez, el minPulse y maxPulse son utilizados para definir el pulso mínimo y máximo que se dan al ángulo mínimo y máximo de cada servomotor. Se ha realizado una evaluación de los pulsos mínimo y máximo de los servomotores MG996R, obteníendose que para 0º el pulso mínimo es 600 y para 180º el pulso máximo sea 2400. Si no fuera así, el servo realizaría un recorrido menor, no cumpliéndose la totalidad de movimiento de 180º.

4.1. Movimiento de la estructura mediante Python y Arduino

En el siguiente código Python, junto al código en Arduino, se realiza una cinemática directa, conectándose a Arduino y llevando cada articulación a la posición indicada. En este apartado se realiza control sólo sobre la estructura física.

Tras añadir las librerías utilizadas, se conecta a Arduino mediante:

```
usbport = '/dev/ttyACM1'
s = serial.Serial(usbport, 9600, timeout=1)
```

Se utilizan funciones para enviar a Arduino el servo a mover y el ángulo a mover.

```
def move(servo, ang):
    s.write(chr(255))
    s.write(chr(servo))
    s.write(chr(ang))
```

Posteriormente, se realizan las inicializaciones de todos los servos, llevándolos a su posición inicial.

```
##ARTICUALCION 1
move(1, 90)
```

En el ejemplo mencionado, vemos como se envía con la función move el servo número 1 y el ángulo al que debe moverse, 90º.

El movimiento de cada articulación desde Python será basado en:

```
    if (opcion == 1):
        time.sleep(1)
        print (")
        print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA ARTICULACION 1')
        angg = input()

        if (0 <= angg <= 180):
            move(1, angg)
```

```
        time.sleep(2)


    else:
        print ('')
        print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')
```

Se introducirá la articulación a mover por pantalla y posteriormente el ángulo al que se desea llevar la articulación. Además existe la opción número “9” para llevar las articulaciones a las posiciones iniciales, y la opción número “0” para salir de la simulación.

La articulación de la pinza, número “5”, irá de 0 a 40 grados, siendo 0 grados la pinza cerrada y 40 grados la pinza abierta.

```
  elif (opcion == 5):
      time.sleep(1)
      print ('')
      print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA ARTICULACION 5')
      print ('LIMITES DE LA PINZA: .0=CERRADA. .40=ABIERTA.')
      angg = input()

      if (0 <= angg <= 40):
          move(5, (360-(3*angg))/2)
# EQUIVALENCIA DEL ANGULO ESTABLECIDO EN EL CODIGO (0 A 40) Y EL
# FIJADO EN EL SERVOMOTOR (180 A 120)
          time.sleep(2)


      else:
          print ('')
              print ('EL GIRO DEL SERVO ESTA LIMITADO A 40 GRADOS'
```

4.2. Movimiento de la estructura y de un modelo en 3D con piezas puras V-REP.

En este objetivo se ha logrado crear un modelo que simula el brazo mediante piezas puras de V-REP y donde se mueve al mismo tiempo el brazo físico. Los códigos utilizados, además del código Arduino, son los códigos en Python utilizados también para el siguiente apartado/objetivo "4.3 Movimiento de la estructura y su representación en V-REP".

4.3. Movimiento de la estructura y su representación en V-REP.

En este caso, se ha conseguido recrear un modelo 3D en SketchUp que representa al modelo físico real. El código en Python realiza un control de cinemática directa y controla el modelo real conectado a la placa Arduino y su simulación en V-REP al mismo tiempo.

Se realiza una conexión a V-REP, mediante los códigos de API remota usados en Python mediante [15]:

```
try:
import vrep
```

```
#### FUNCIONES V-REP ####

def conectarVREP():
        vrep.simxFinish(-1) # CERRAR TODAS LAS CONEXIONES ABIERTAS
        clientID = vrep.simxStart('127.0.0.1',19999,True,True,5000,5)
        # CONECTAR A V-REP

        if (clientID != -1):
            print ('CONECTADO AL SERVIDOR DE API REMOTA')
            vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot_wait)
            vrep.simxSynchronous(clientID,True) # MODO SINCRONO
            return clientID

        else:
            print ('ERROR: PROBLEMA AL CONECTAR CON EL SERVIDOR DE API')
            print ('REMOTA')
            sys.exit(0);
```

```
def iniciarSim(clientID):
        vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot)

def pararSim(clientID):
        vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot_wait)

def desconectarVREP(clientID):
        vrep.simxFinish(clientID)
        print ('SESION CON V-REP FINALIZADA')
```

```
clientID=conectarVREP()
```

De esta manera se abre y cierra una conexión con V-REP usando estas funciones de la API remota en Python. En el capítulo 5 “Simulación en V-REP” del presente Trabajo Fin de Grado se encuentra una descripción detallada de la conexión entre Python y V-REP mediante la API remota de V-REP.

Además, existe una inicialización de comunicación con los *handlers* de las articulaciones en V-REP.

```
## ARTICULACION 1
ret,joint1_handler = vrep.simxGetObjectHandle(clientID,"Joint1",vrep.simx_opmode_oneshot_wait)
ret,joint1 = vrep.simxGetJointPosition(clientID,joint1_handler,vrep.simx_opmode_streaming)
move(1, 90)
```

Para el movimiento de las articulaciones, tanto en V-REP como en Python se utiliza funciones para recibir la posición de cada articulación y para enviar la posición a cada articulación.

Para recibir la posición de cada articulación del modelo en V-REP se utiliza:

```
        print ('')
        print ('POSICION EN GRADOS DE LAS ARTICULACIONES:')
        ret,joint1 = vrep.simxGetJointPosition(clientID,joint1_handler,vrep.simx_opmode_buffer)
        print ('1: ' + str(int(joint1*(180/pi))))
```

Como vemos, hay una función concreta de la API remota utilizada en Python para obtener la posición de una articulación, esta es:

- vrep.simxGetJointPosition [15]: Utilizada para obtener la posición de una articulación en V-REP .

Para enviar la posición, en ángulos, a la que debe llegar cada articulación se utiliza el código:

```
print ('')
print ('INDIQUE LA ARTICULACION A MOVER.')
opcion = input()

if (opcion == 1):
    time.sleep(1)
    print ('')
     print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA')
    print ('ARTICULACION 1')
    angg = input()

    if (0 <= angg <= 180):
      angr = (angg*(pi/180))
      ret =vrep.simxSetJointPosition(clientID,joint1_handler,angr,vrep.simx_opmode_oneshot)
      move(1, angg)
      time.sleep(2)

    else:
      print ('')
      print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')
```

Como se aprecia, se utiliza otra función de la API remota, en este caso, para enviar una posición a la "joint1" del modelo en V-REP.

- vrep.simxSetJointPosition [15]: Utilizada para dar la posición a una articulación en V-REP.

Además, dado que el giro del servomotor sólo puede variar de entre 0º y 180º, únicamente se permitirá introducir por pantalla un valor entre estos límites. La introducción de un valor fuera de este rango hará que se solicite de nuevo un valor al usuario.

Al igual que en el objetivo anterior, existe una opción para llevar todas las articulaciones a las posiciones iniciales. Esta opción está definida por:

```
        elif (opcion == 9):
            time.sleep(1)
            print (")
            print ('LLEVANDO A LAS ARTICULACIONES A SUS POSICIONES')
            print ('INICIALES')
            ret = vrep.simxSetJointPosition(clientID,joint1_handler,pi/2,vrep.simx_opmode_oneshot)
            move(1, 90)
            time.sleep(1)
            ret = vrep.simxSetJointPosition(clientID,joint2_handler,pi/2,vrep.simx_opmode_oneshot)
            move(2, 90)
            time.sleep(1)
            ret = vrep.simxSetJointPosition(clientID,joint3_handler,pi/2,vrep.simx_opmode_oneshot)
            move(3, 90)
            time.sleep(1)
            ret = vrep.simxSetJointPosition(clientID,joint4_handler,pi/2,vrep.simx_opmode_oneshot)
            move(4, 90)
            time.sleep(1)
            ret = vrep.simxSetJointPosition(clientID,joint5_handler,0,vrep.simx_opmode_oneshot)
            move(5, 180)
            time.sleep(2)
```

Se observa cómo se lleva tanto en V-REP como en el modelo físico cada articulación a su posición inicial.

Finalmente para finalizar y desconectar con V-REP se utilizan las funciones definidas al comienzo del código, que mediante el uso de funciones de la API remota, se encargan de desconectar Python de V-REP:

```
pararSim(clientID)

desconectarVREP(clientID)
```

4.4. Movimiento del modelo real y su representación con “sliders” desde V-REP.

En este caso se utilizan tres códigos distintos. El código en Arduino será diferente al resto de los casos, utilizando la librería “Servo.h” ya que el control de velocidad lo hace el usuario mediante el movimiento de los *sliders*. Se recogen los datos y posiciones de cada articulación de V-REP mediante un código Python. A su vez, en V-REP existe un código implementado, en LUA, que se ocupa de dar posiciones a las articulaciones mediante *sliders* o botones deslizantes. De esta manera, en este objetivo logrado sería desde V-REP donde se envían las posiciones de las articulaciones y Python funcionaría como puente entre V-REP y Arduino.

En el código en Python, tras realizar las necesarias conexiones a V-REP y sus respectivas inicializaciones, cuales son como en el apartado anterior, se recibe la posición en la que se encuentra cada articulación del modelo en V-REP y se envía esta posición al servo relacionado del modelo físico conectado a la placa Arduino.

```
        ret,joint1 = vrep.simxGetJointPosition(clientID,joint1_handler,vrep.simx_opmode_buffer)
        print ('1: ' + str(int(joint1*(180/pi))))
        move(1, int(joint1*(180/pi)))
```

Por otro lado, existe un código en LUA utilizado para dar el ángulo a cada posición mediante los *sliders* creados en V-REP.

```
if (simGetScriptExecutionCount()==0) then
    --Codigo inicializacion
    ctrlID=simGetUIHandle("UI")
    sliderID={4,6,8,10,12}
end
```

En esta parte se recogen la posición del *handle* de cada *slider*, mediante el uso de la función "simGetUIHandle” de la API de V-REP [14]. De esta manera se sabrá la posición en el panel de cada botón deslizante que enviará la posición a cada articulación en V-REP.

El siguiente extracto de código es utilizado para determinar el objeto que hace referencia a cada articulación en V-REP. Se utiliza la función "simGetObjectHandle" [14].

```
qID={}
for i=1,5,1 do
qID[i]=simGetObjectHandle('Joint' .. i)
end
```

Posteriormente, se procede dentro del *main* a mostrar por pantalla la posición de cada *slider*, que tras una conversión será desde 0 a 180, referenciándose al rango en ángulos de movimientos permitidos para cada articulación

```
print('SLIDERS')
for i=1,#sliderID,1 do
    slider_i = simGetUISlider(ctrlID,sliderID[i])
   -- print('slider' .. i .. '=' .. slider_i) Mostraremos solamente un angulo entre 0 y 180 grados
    slider_iconv = (slider_i*0.18)
    print ('slider' .. i .. '=' .. slider_iconv)
       end
```

También se mostrará la posición en cada instante de cada articulación. Para este paso, la función usada para obtención de la posición de una articulación tiene una forma muy parecida a la función en Python. En este caso es "simGetJointPosition" [14].

```
print('Posiciones de las articulaciones')

for i=1,#qID,1 do
    joint_i=simGetJointPosition(qID[i])
    joint_iconv=((joint_i/(2*math.pi))*360)
    print('La posicion de la joint' .. i .. ' es ' .. joint_iconv)
end
```

Finalmente, se obtendrá la posición de cada *slider*, que va de 0 a 1000 pero se realiza una conversión para que sea de 0 a 180 (en la pinza el rango será de 0 a 40), y se le adjudicará a cada articulación mediante la función "simSetJointPosition" [14] utilizada en el extracto de código:

```
for i=1,#sliderID,1 do
    slider_i = simGetUISlider(ctrlID,sliderID[1])
    slider_iconv = (slider_i*0.18)
    slider_iconvrad = ((slider_iconv/180)*(math.pi))
   for i=1,#qID,1 do
       result=simSetJointPosition(qID[1],slider_iconvrad)
    end
end
```

En V-REP se crean los *sliders* de manera visual, para posteriormente programar su funcionamiento en el código. Estos botones deslizantes tienen una forma visual como la de un panel que puede ser utilizado por el usuario:

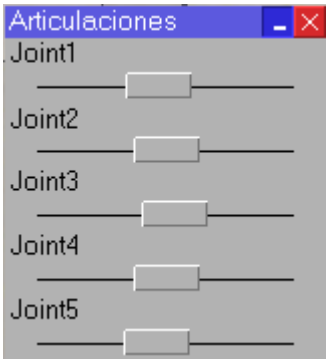

Figura 39. *Sliders* en V-REP

5

5. Simulación en V-REP

Una simulación en V-REP tiene una serie de pasos previos. La interfaz ofrece diversos modelos robóticos que pueden ser utilizados como ejemplos pero también admite la creación de un modelo propio. Este modelo puede ser creado a partir de piezas *"primitive shapes"*, cuales son tratadas como piezas puras dentro del programa y son las más recomendadas y estables para simulaciones dinámicas.

Figura 40. Pieza pura en V-REP [13]

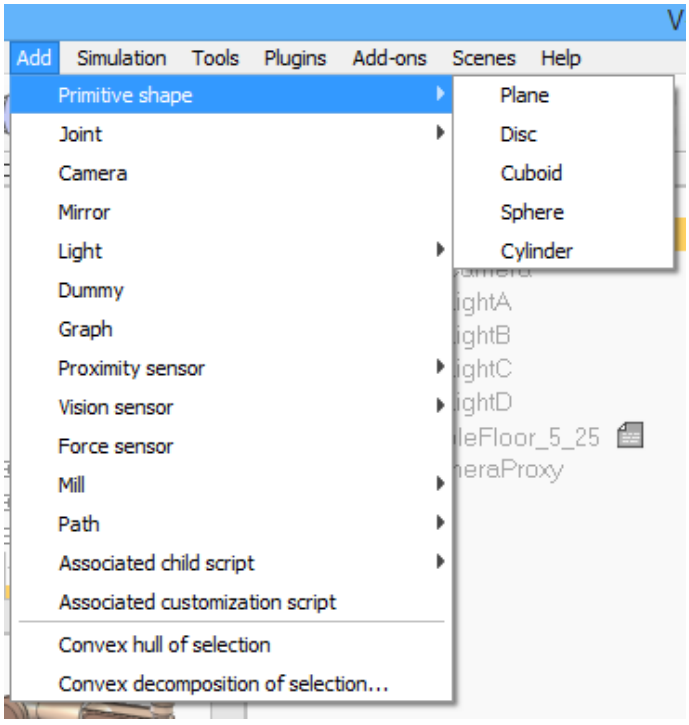

Figura 41. Añadir una pieza pura en V-REP

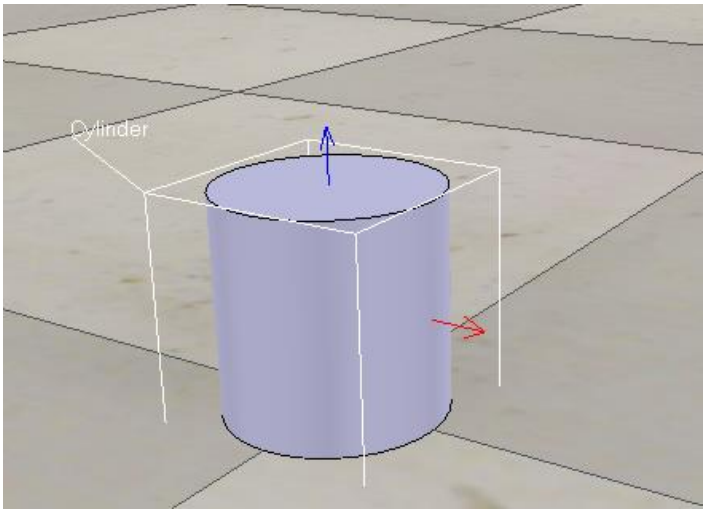

Figura 42. Visualización de una pieza pura en V-REP

En cambio, el modelo también puede ser creado a partir de piezas importadas desde otros programas, como por ejemplo en ".stl". Este tipo de piezas son las más inestables a la hora de simular dinámicamente debido a que tienen un mayor "peso". También, en una simulación, aparte de poder ocasionar inestabilidad sucederá que la velocidad de trabajo y rendimiento será menor debido a un mayor número de cálculos.

Figura 43. Pieza importada en V-REP [13]

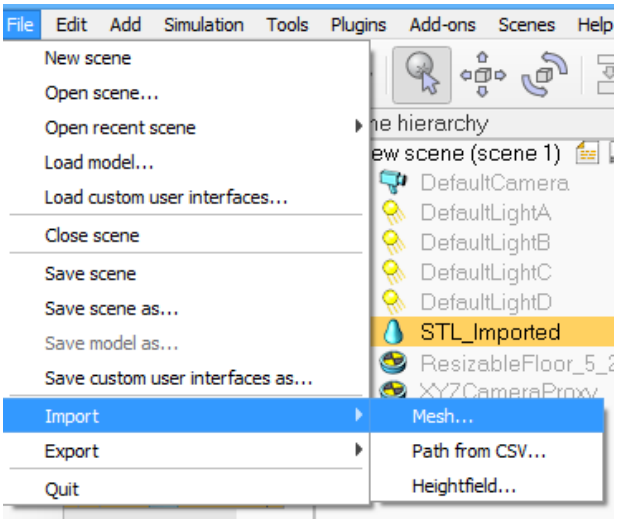

Figura 44. Importación de una pieza en V-REP

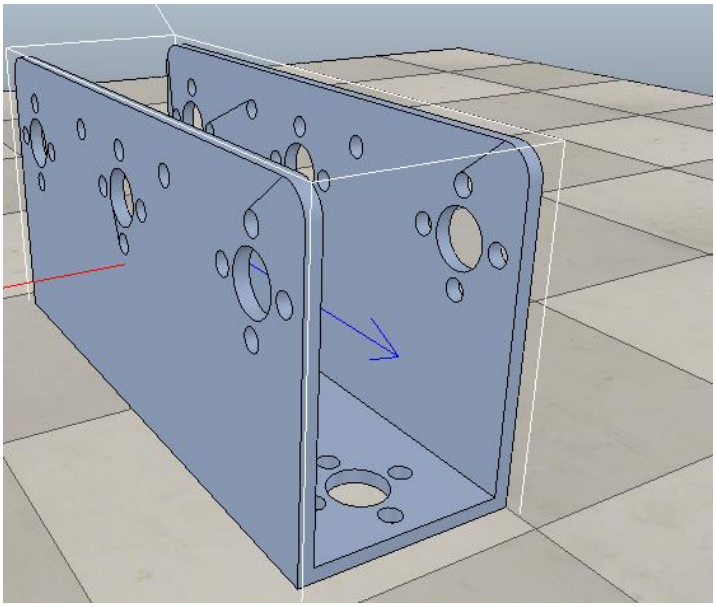

Figura 45. Visualización de una pieza ".stl" importada en V-REP

Tras la creación o importación de cada una de las piezas hay que posicionarlas y orientarlas como el modelo a crear necesite. Para eso se utiliza la ventana en V-REP *"Object/Item position/orientation"*.

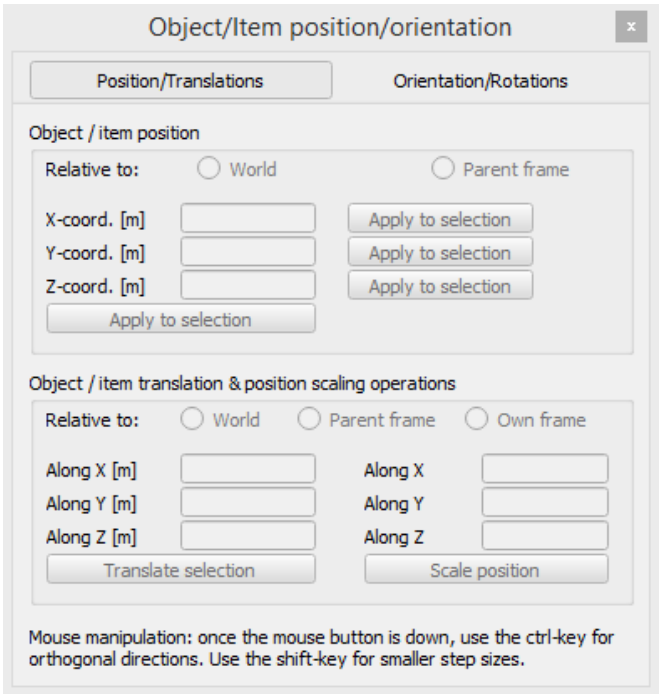

Figura 46. Ventana "Object/Item position/orientation"

Posterior a la colocación de cada una de las piezas se obtendrá la estructura deseada. Pero para poder realizar una simulación será necesario añadir articulaciones, *Joints*. Las articulaciones en V-REP pueden ser de revolución, prismáticas o esféricas, también tiene diferentes modos de funcionamiento. Se posicionan y se orientan dentro de la estructura del brazo de la misma manera que se hace con las piezas.

Para que una articulación sepa entre qué dos piezas se encuentra se utiliza un sistema de jerarquía. Se selecciona una articulación en primer lugar, que actuará como objeto "hijo", y luego se selecciona la pieza deseada de la que depende la articulación, para de esta manera hacer el último objeto seleccionado el objeto "padre".

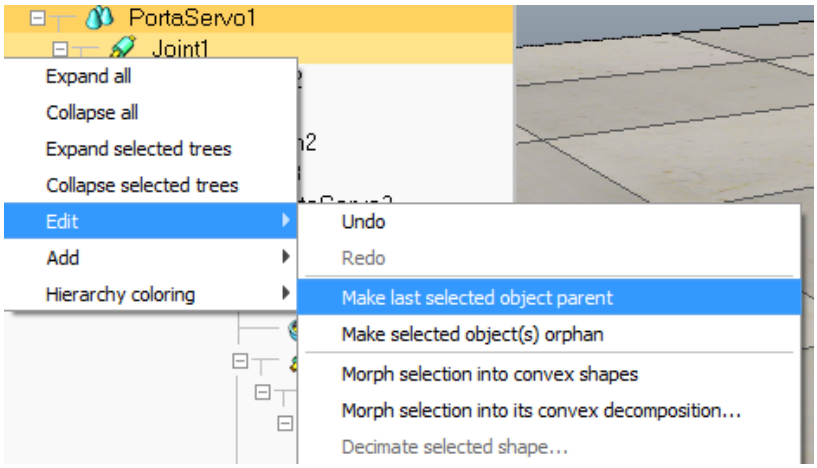

Figura 47. Establecer una jerarquía en V-REP

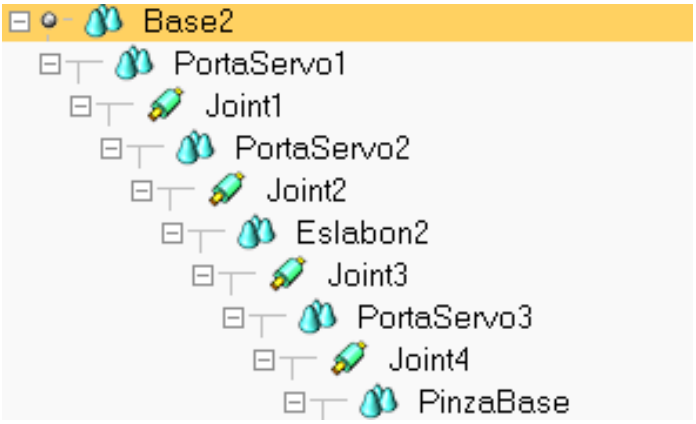

Figura 48. Visualización de una jerarquía en V-REP

Realizada ya toda la jerarquía, obtendríamos que cada pieza, ligadas desde la base hasta la pinza, tendrá su correspondiente articulación donde fuese necesario. De esta manera, si se mueve la articulación de la base, que es la primera, dado que las demás articulaciones y piezas dependen de la primera articulación, se movería el resto del brazo según se mueva dicha articulación.

Finalmente, obtendríamos un modelo preparado para implementarle movimientos dinámicos. Hay que tener en cuenta que existe una gran variedad de características para los modelos de V-REP que se utilizarán en caso de que sea necesario.

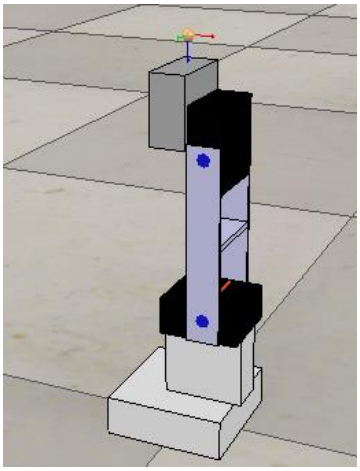

Figura 49. Modelo creado a partir de piezas puras en V-REP

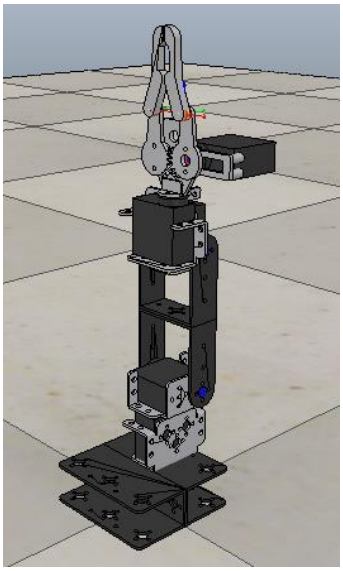

Figura 50. Modelo creado a partir de piezas ".stl" importadas en V-REP

5.1. API remota de V-REP con Python

En el presente Trabajo Fin de Grado, en todos los casos que se utiliza V-REP, se realiza una conexión con Python. Esta conexión es posible gracias al uso de funciones de la API remota de V-REP. En este caso las funciones de conexión remota están orientadas para el uso de Python.

El proceso de conexión comienza con la importación en el código de un archivo llamado "vrep". Este mismo archivo lo proporciona V-REP y simplemente hay que tenerlo en el mismo directorio que el código Python que se vaya a utilizar.

```
try:
import vrep
```

En el código Python se establecen las conexiones a V-REP mediante el uso de funciones de la API remota de V-REP para Python. En primer lugar, cierra todas las posibles conexiones abiertas mediante la función "vrep.simxFinish" en V-REP y así se garantiza una conexión correcta mediante el uso de la "ip" y del puerto a utilizar mediante "vrep-simxStart".

```
def conectarVREP():
        vrep.simxFinish(-1) # CERRAR TODAS LAS CONEXIONES ABIERTAS
        clientID = vrep.simxStart('127.0.0.1',19999,True,True,5000,5)
        # CONECTAR A V-REP
```

Los demás datos son:

- True: en caso de ser true, garantiza un bloqueo de la función hasta que se conecte.
- True: en caso de ser true, no se conecta de nuevo una vez se desconecte.
- 5000: al ser positivo se refiere al tiempo en milisegundos para realizar la primera conexión.
- 5: número recomendado que indica cada cuanto se envía y se reciben paquetes de datos.

Si se conecta será:

```
if (clientID != -1):
    print ('CONECTADO AL SERVIDOR DE API REMOTA')
    vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot_wait)
    vrep.simxSynchronous(clientID,True) # MODO SINCRONO
    return clientID
```

Como se aprecia, se comienza la simulación una vez se garantice una correcta conexión mediante el uso de la función "vrep.simxStartSimulation". En esta función se describe el tipo de operación, que será de un disparo, un *trigger*. Por otro lado, "vrep.simxSynchronous" es utilizada para habilitar un modo síncrono en la conexión.

Si el clientID es igual a -1, dará error al conectar a la API remota.

```
else:
    print ('ERROR: PROBLEMA AL CONECTAR CON EL SERVIDOR DE API')
    print ('REMOTA')
    sys.exit(0);
```

A continuación, se definen:

- iniciarSim: para iniciar la conexión, uso de la función vrep.simxStartSimulation.
- pararSim: para parar la conexión, uso de la función vrep.simxStopSimulation.
- desconectarVREP: para desconectar y finalizar por completo la conexión, uso de vrep.simxFinish.

```
def iniciarSim(clientID):
    vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot)

def pararSim(clientID):
    vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot_wait)
```

```
def desconectarVREP(clientID):
        vrep.simxFinish(clientID)
        print ('SESION CON V-REP FINALIZADA')
```

Se conecta a VREP mediante el clientID.

```
clientID=conectarVREP()
```

Finalmente, para parar la simulación, se introduce al final del código:

```
pararSim(clientID)


desconectarVREP(clientID)
```

Posteriormente, se realizan las inicializaciones y la obtención o envío de los ángulos de cada una de las articulaciones. Este proceso se puede ver tanto en el capítulo 4 “Programación” como en el Anexo I “Códigos implementados”.

6

6. Experimentos y pruebas del sistema

En este capítulo se realizará una exposición de las diferentes pruebas realizadas con distintos tipos de códigos, así como la reacción del modelo físico y el modelo virtual según cada uno de los códigos implementados, ya que existe un proceso de resolución de errores y mejoras técnicas realizadas a lo largo del proyecto que han ayudado a obtener unos resultados de respuesta del brazo más claros y mejores.

6.1. Experimentos y pruebas en la estructura física

El primero de los cambios realizados en el proyecto fue el número de grados de libertad disponibles en brazo robótico utilizado. Como comentábamos en el capítulo 2 "Estudio previo y evolución del proyecto" de la presente memoria, el número de articulaciones soportadas por la estructura del brazo era de 6 articulaciones aunque se llega a utilizar un brazo de sólo 5 articulaciones. La figura 51 muestra la forma del brazo con 6 grados de libertad mientras que en la figura 52 se muestra el brazo utilizado en el proyecto con 5 grados de libertad:

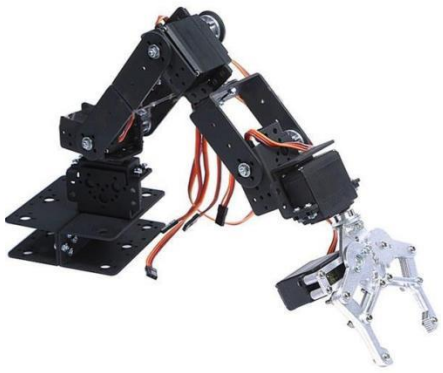

Figura 51. Estructura del brazo con 6 gdl

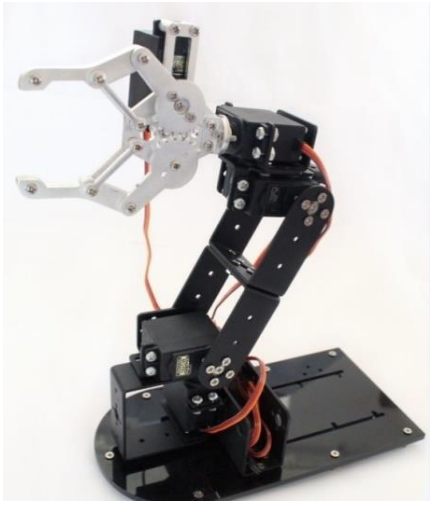

Figura 52. Manipulador antropomórfico de 5 gdl

Como se puede visualizar en la figura 51, el brazo es más largo. Esto conllevaría a que los servos tuvieran una mayor carga para realizar los respectivos movimientos del brazo.

Aunque viendo el peso de la estructura y la capacidad de los servos, la idea de utilizar 6 gdl era viable, tras un primer montaje, se podía apreciar como el servomotor de la articulación número 2, partiendo desde la base, no soportaba en ningún momento el peso del brazo. Incluso, tras realizar un intento de mover un rango en ángulos muy pequeño, el servomotor utilizado en dicha posición nunca pudo realizar el movimiento señalado.

Posteriormente, viendo las consecuencias de utilizar una estructura con 6 gdl, se optó por el modelo representado de la figura 52, con 5 grados de libertad. En la posición de la articulación número 2 se procedió a colocar un servomotor nuevo del mismo modelo, para garantizar un estado óptimo de todas las articulaciones.

A partir de este momento, ninguna de las articulaciones del brazo físico dio ningún problema más, dando la oportunidad de avanzar a la implementación de los códigos en Python para el movimiento de la estructura.

6.2. Experimentos y pruebas en el código Arduino

En el presente Trabajo Fin de Grado, el código implementado en Arduino es para todos los casos de control el mismo menos en el control desde V-REP, que utilizará una librería diferente la "Servo.h". Como señalábamos en el capítulo 4 "Programación", se utiliza la librería "VarSpeedServo.h" para el control de la velocidad del movimiento de cada servomotor. Con el uso de esta librería se pueden dar velocidades diferentes a cada servomotor.

Pero, en un primer lugar, no se utilizaba dicha librería, por lo que todos los servomotores se movían a la máxima velocidad, ya que así venía programado por defecto en la librería "Servo.h". Este punto ocasionaba diversos problemas a la hora de realizar movimiento de la estructura física.

Uno de los problemas que ocasionaba es que si alguno de los servomotores se encontraba en una posición inicial y quería ir a una posición cercana a su límite, sin llegar a ser el límite, debido a la velocidad del servomotor y al peso que movía, el resultado obtenido era que el servomotor, en ocasiones, superaba el ángulo deseado debido a la inercia del movimiento.

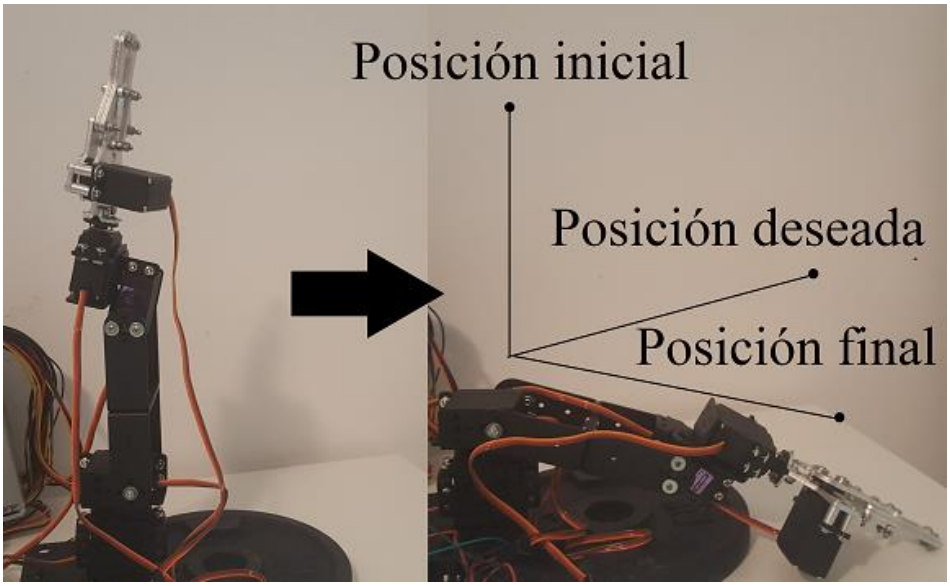

Figura 53. Variación de la posición final debido a la inercia del movimiento del brazo

También podía suceder que el servomotor llegara a la posición indiciada, pero que antes de quedarse quieto sin realizar ningún movimiento en dicha posición, hiciera un movimiento de tipo vaivén o resorte. A efecto visual, sucedía como si se realizara un amortiguamiento del movimiento. Esto se debía a la velocidad e inercia que llevaba el brazo y a la parada sin desaceleración.

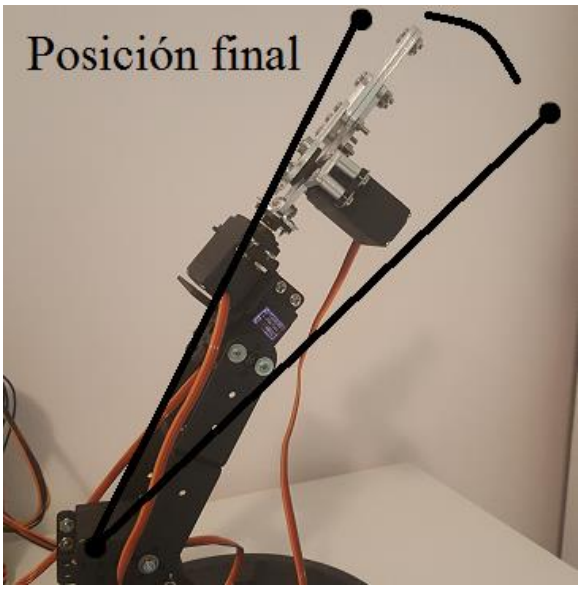

Figura 54. Variación tipo vaivén o resorte debido a la inercia del movimiento del brazo

Cabe decir que cuando se realizaban los movimientos parecía que el brazo físico y el virtual se movían al mismo tiempo y a tiempo real. Pero este dato es erróneo en ambos casos. No se mueven ni al mismo tiempo ni el modelo en V-REP se mueve a tiempo real.

Incluso aunque pareciese que el movimiento de los servomotores a máxima velocidad satisface la necesidad de ver el modelo real y virtual realizando movimientos al mismo tiempo, este argumento no sería suficiente, ya que el movimiento a máxima velocidad, como hemos visto anteriormente, puede dañar a cada uno de los servomotores.

Veríamos entonces que esta opción, mover los servos a máxima velocidad, no sería viable porque dañaba a los servomotores, además de ocasionar problemas en el movimiento del brazo. Dado este punto, se realizó la implementación de un código en Arduino que facilitase el control de velocidad en cada uno de los servomotores, tal y como se ha hecho en la presente memoria.

Con librería "VarSpeedServo.h", al darle a un servomotor valor 1 a la velocidad iría a mínima velocidad y al darle valor 255 se movería máxima velocidad. Dentro de este rango, el aumento del valor numérico supondría un aumento de la velocidad de movimiento del servomotor. Además, cabe destacar que si se da el valor 0, o no se da ningún valor (valor por defecto sin usar la librería mencionada), el servomotor se movería a máxima velocidad.

Dado que la articulación 1 es de rotación vertical sobre el eje z como se puede apreciar en la figura 10 y en la figura 11 del capítulo 2 "Estudio previo y evolución del proyecto", se puede ver que la articulación 2 tiene que soportar el peso del brazo entero como se ven en las figuras 52, 53 y 54. La facilidad de dar diferentes velocidades a los servomotores de cada articulación hizo que consideráramos darle la mitad de velocidad de movimiento al servomotor localizado en la articulación número 2. De esta manera se garantiza completamente un estado óptimo del servomotor situado en dicha articulación.

6.3. Experimentos y pruebas en el código Python

El código Python sirve tanto para controlar los modelos real y virtual como para obtener información de uno de los modelos y enviarle al otro modelo dicha información, en este caso, las posiciones de cada articulación.

Dado esto, los códigos implementados en Python también han variado a medida que avanzaba el proyecto parar obtener un mejor resultado y una mejor interacción con el usuario. Como se señalaba en el capítulo 4 "Programación", se añade finalmente por consola un panel de usuario donde se pide la articulación a mover (de 1 a 5) y el ángulo al que se desea mover la articulación (de 0º a 180º). También existe la posibilidad de introducir un "9" para llevar todas las articulaciones a posiciones iniciales, o introducir un "0" para salir de la simulación.

Figura 55. Panel de usuario en Python

6.4. Experimentos y pruebas en V-REP

En V-REP, tras finalizar los modelos utilizados, como se explica en el capítulo 6 "Simulación en V-REP", hay que realizar una serie de ajustes para que los resultados obtenidos en el movimiento del modelo en V-REP sean los deseados.

Uno de los factores que hay que tener en cuenta son las inicializaciones de las articulaciones (*joints*) en V-REP. Las articulaciones en V-REP deben estar inicializadas tal y como están inicializadas en el modelo físico. De esta manera, los movimientos serán acordes tanto en el modelo físico como en el modelo virtual

Figura 56. Posiciones iniciales de las articulaciones en el modelo físico

En la figura 56 se puede ver como son las inicializaciones de las articulaciones del modelo físico. Dado esto, se ve como en el modelo virtual se inicializa de la misma manera en la figura 57.

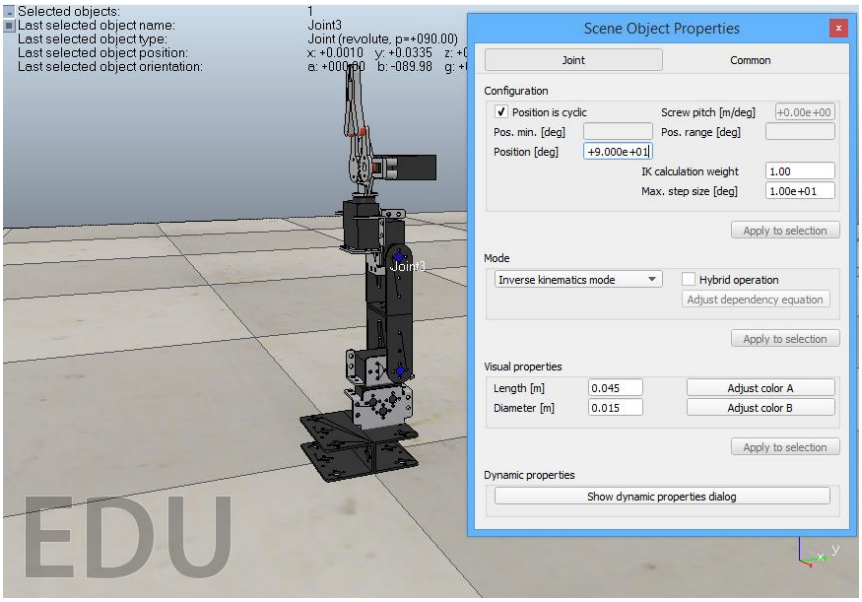

Figura 57. “Joint3” inicializada a 90º en V-REP

Pero en un primer momento todas las articulaciones estaban orientadas erróneamente en el modelo virtual. Un ejemplo, es la “joint3” que se visualiza en la figura 58. Debido a esto, el movimiento en el modelo físico no era representado por el movimiento del modelo virtual.

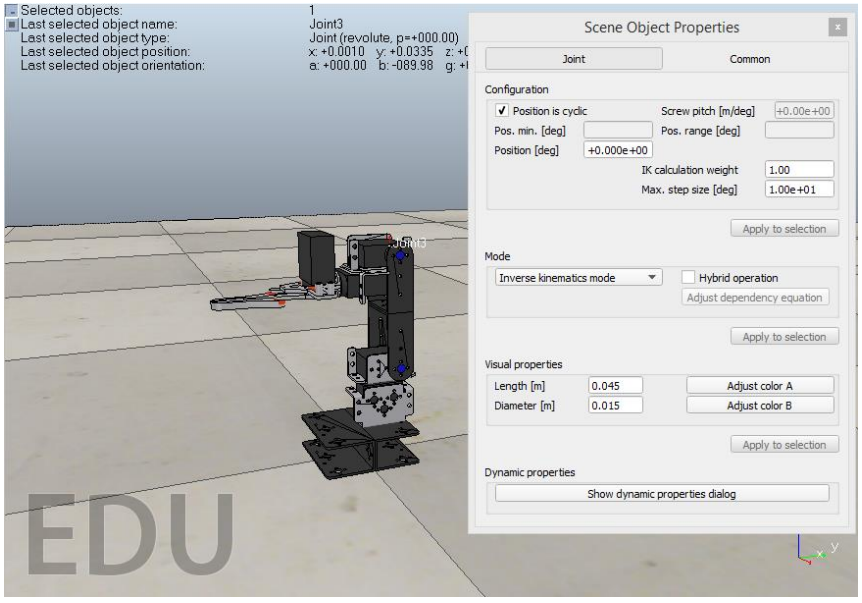

Figura 58. “Joint3” inicializada erróneamente por defecto a 0º en V-REP

6.5. Experimentos y pruebas con potenciómetro para control de posición de servomotores

En este caso, se utilizó un potenciómetro para calcular los valores de mínimo pulso y máximo pulso necesarios para mover los servomotores en un rango completo de 0º a 180º. El potenciómetro variaba entre los valores 0 y 1023. Dicho 0 correspondía a 0º y dicho valor 1023 correspondía a 180º.

Dado esto, el potenciómetro hace girar el servomotor correspondiente. Es entonces cuando, según los pulsos mínimo y máximo utilizados, se comprueba si llega a los valores marcados por el potenciómetro y por lo tanto si hace el recorrido completo entre 0º y 180º.

pruebaservo §

```
#include <Servo.h>

Servo servo;


const int pinPot = 0;
const int pinServo = 2;
const int pulsoMin = 600;
const int pulsoMax = 2400;

int ang;
int valor;


void setup() {
  Serial.begin(9600);
  servo.attach(pinServo, pulsoMin, pulsoMax);
}

void loop() {
  valor = analogRead(pinPot);
  Serial.println(valor);
  ang = map(valor, 0, 1023, 0, 180);
  Serial.println(ang);
  servo.write(ang);
  delay(20);
}
```

Figura 59.Código en Arduino para comprobación de rango mediante potenciómetro

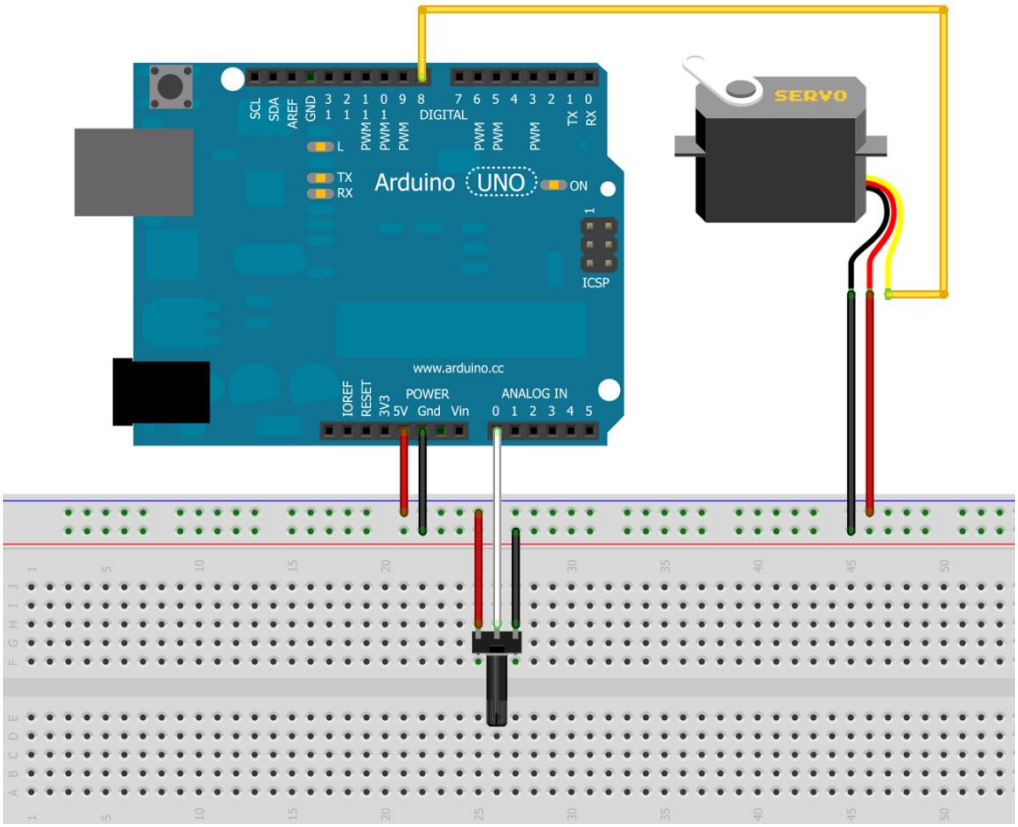

Figura 60. Conexionado Arduino con potenciómetro y servomotor

7

7. Presupuesto

El siguiente presupuesto recoge todos los gastos del proyecto con impuestos y gastos de envío incluidos.

Artículo	Precio U. (€)	Cantidad	Precio T. (€)
Microcontrolador Arduino Uno	30	1	30
Estructura Brazo Robótico 6GDL Variable	28,99	1	28,99
Servomotor MG996R	8,70	5	43,5
Pack Protoboard y 65xCable Jumper Arduino	12,90	1	12,90
ATX Power Suply 500 W	20	1	20
Software Ardunio Genuino*	0	1	0
Software Python Spyder*	0	1	0
Software V-REP EDU*	0	1	0
		Total	135,39 €

Tabla 2. Presupuesto del presente Trabajo Fin de Grado

*Los softwares utilizados en el presente Trabajo Fin de Grado son todos de licencia gratuita en sus respectivas versiones.

8

8. Conclusions

This final of degree Project is finished after a large academic course. It has been a useful time which in order to affront diverse situations that the project gave, it helped to acquire new knowledges and to know how to solve new problems that appeared in different phases when the project advanced. For us is new the use of software like Arduino and V-REP, and the programming in Python, Arduino and LUA. These points assure to learn new abilities in programming languages and the use of new applications and interfaces like V-REP (Virtual Robot Experimentation Platform).

On another hand, the difficult that appears after changes in the situations and plans for the project (like eliminate parts how the elaboration of the model in a 3D printer cause of the complexity and the economic and temporary spending, like to add to the project the use of V-REP interface, etc.) have been solved. Furthermore, in a project like that, the user or investigator needs to be ready to visualize and to solve this type of situations with best possible alternative way.

A work line where we can see the different steps and the improvement of the project with the time defines this final work.

As a first point, we need to acquire new knowledge, how we said, in Python, Arduino and V-REP, software and programming languages that we never used. This part, the first one, even if it is the worst in order to see the results, it is one of the most important parts, because it is needed to know how to program and to have the idea of what the robot will do with the programming, the robot work programmed in the control.

Secondly, it is time to think about the structure and the creation of 3D model in the V-REP interface. Done that, the programming part is the hardest part of the project. To implement a code to communicate Python, Arduino and V-REP at the same time, and move the articulations of the robot has to be correct in every line of the code to avoid errors that can produce interferences in the move of the industrial manipulator. But although, this part is hard, when it is finished, it shows the first parts of the results that we expected.

Finally, it is time to set up the movement of the robot, the control, and the velocity of the articulations, even, it is time to set up a control panel for the user. How we used random pieces in ".stl" format, V-REP cannot work in the best way, showing sometimes interferences in the movement of the robot. But it is caused because of the type of the piece. With that, we finalized the project and we can say that we achieved our objectives, obtaining the final result that we can control at the same time a 3D model recreated in a virtual simulator and a real model, using three different types of interfaces interconnected.

Furthermore, there is a work line that gives the possibility to continue with project. It seems that can be controlled and programmed to generate and follow complex path, and the design, in another point, of an inverse kinematic that help this type of movement. Even, it is possible to create object in V-REP to work like real obstacles that the structure can find in a real map.

In conclusion, highlight that this is a project that is based and realized, to a large degree, in the use of programming. But with the advance of the project we can appreciate the robotic results that we expected. So that it is a final work of degree of great interests in many aspects.

9

9. Bibliografía

La siguiente reseña bibliográfica se encuentra ordenada alfabéticamente por apellido del autor. En el caso de existir más de un autor se tendrá en cuenta el primero de los mismos para su ordenación. A su vez, los contenidos y recursos "online" serán situados de la misma manera pero teniendo en cuenta la institución que realiza el documento informativo en el caso de que no exista autor.

[1] Acosta, L. 2014. Sistemas Robotizados.

[2] Arduino. 2016. Disponible en la URL: https://www.arduino.cc/

[3] Entorno de simulación de Autodesk. Autodesk Circuits. 2016. Disponible en la URL: https://circuits.io/

[4] Dinwiddie, K. 2014. Basic Robotics. Cengage Learning.

[5] Primer diseño de Robot Humanoide ASIMO. Honda. 2016. Disponible en la URL: http://asimo.honda.com/

[6] Definición de robot industrial de manipulación. International Federation of Robotics. 2016. Disponible en la URL: http://www.ifr.org/industrial-robots/

[7] Definición de robot industrial. ISO 8373:2012. Disponible en la URL: https://www.iso.org/obp/ui/#iso:std:iso:8373:ed-2:v1:en

[8] Ollero, A. 2001. Robótica: manipuladores y robots móviles. Marcombo.

[9] Programming Language Python. Python. 2016. Disponible en la URL: https://www.python.org/

[10] Python Tutorial. 2016. Disponible en la URL: https://docs.python.org/2/tutorial/

[11] SketchUp. 2016. Disponible en la URL: http://www.sketchup.com/

[12] U.S. Congress. April 1984. Manufacturing Automation: Employment, Education and the Workplace. Washington, D.C. Office of Technology AssessmentComputerized.

[13] V-REP. Virtual Robot Experimentation Platform. 2016. Disponible en la URL: http://www.coppeliarobotics.com/

[14] API Functions. Descripción de funciones utilizadas en LUA para V-REP. V-REP. 2016 Disponible en la URL: http://www.coppeliarobotics.com/helpFiles/en/apiFunctions.htm

[15] REMOTE API Functions (Python). Descripción de las funciones remotas para conexión entre Python y V-REP. V-REP. 2016. Disponible en URL: http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm

[16] Librería "VarSpeedServo.h" utilizada en Arduino. Disponible en la URL: https://github.com/netlabtoolkit/VarSpeedServo

[17] Descripción de "ASIMO". Wikipedia. 2016. Disponible en la URL: https://es.wikipedia.org/wiki/ASIMO

[18] Descripción de "Caballero mecánico", Robot de Leonardo Da Vinci. Wikipedia. 2016. Disponible en la URL: https://es.wikipedia.org/wiki/Robot_de_Leonardo

[19] Descripción de "Robótica" y revisión histórica. Wikipedia. 2016. Disponible en la URL: https://es.wikipedia.org/wiki/Rob%C3%B3tica

Anexo I

Anexo I. Códigos implementados

En el presente Anexo se expondrán todos los códigos utilizados en cada uno de los módulos softwares. En los capítulos 4 y 5, "Programación" y "Simulación en V-REP" respectivamente, se realizan explicaciones sobre el código y cada una de las funciones utilizadas

Para cada uno de los objetivos logrados existen diferentes códigos en Arduino, Python y V-REP (LUA). El código en Arduino es general y usado por todos y cada uno de los objetivos, ya que con el código se envían los ángulos a los servomotores, cuales están conectados a los diferentes pines del controlador Arduino.

El código en Arduino sería:

```
#include <VarSpeedServo.h>

// SE INCLUYE LA LIBRERIA <VarSpeedServo.h> PARA DECLARAR LOS SERVOS
// DICHA LIBRERIA PERMITE REGULAR LA VELOCIDAD DE MOVIMIENTO
// DECLARACION DE LOS SERVOS

VarSpeedServo servo1;
VarSpeedServo servo2;
VarSpeedServo servo3;
VarSpeedServo servo4;
VarSpeedServo servo5;

// MINIMO Y MAXIMO PULSO DE UN SERVO PARA IR DE 0º A 180º
```

```
int minPulse = 600;
int maxPulse = 2400;
int userInput[2];
int incomingByte;

int servoNumber;
int angle;
int i;
int ledPin = 13;
int pinState = LOW;

void setup() {
    Serial.begin(9600);

// SE ASIGNA A CADA SERVO DECLARADO UN PIN DEL ARDUINO
    servo1.attach(3, minPulse, maxPulse);
    servo2.attach(5, minPulse, maxPulse);
    servo3.attach(6, minPulse, maxPulse);
    servo4.attach(9, minPulse, maxPulse);
    servo5.attach(10, minPulse, maxPulse);
    pinMode(ledPin, OUTPUT);

// SE ENVIA A LOS SERVOS A SUS POSICIONES INICIALES
    servo1.write(90);
    servo2.write(90);
    servo3.write(90);
    servo4.write(90);
    servo5.write(180);
}

void loop() {
    if (Serial.available() > 2) {
// SE ESPERA RECIBIR MAS DE DOS DATOS
```

```
        incomingByte = Serial.read();
// EL PRIMER DATO RECIBIDO ES PARA DETERMINAR QUE ESTAMOS ANTE
// LA RECEPCIÓN DE UN MENSAJE QUE LLEVA ASOCIADO 2 DATOS
// NUMERO DE SERVO Y ANGULO
        if (incomingByte == 0) {
            for (i=0; i<2; i++) {
                userInput[i] = Serial.read();
                }
                servoNumber = userInput[0];
                angle = userInput[1];

// SE ENVIA AL SERVO UN ANGULO SEGUIDO DE UN NUMERO (VELOCIDAD)
// COMPRENDIDO ENTRE  1 Y 255 (LENTO/RAPIDO, 0 ES IGUAL A MAXIMA
// VELOCIDAD
                switch (servoNumber) {
                case 1:
                    servo1.write(angle, 40, true);
                    break;
                case 2:
                    servo2.write(angle, 20, true);
                    break;
                case 3:
                    servo3.write(angle, 40, true);
                    break;
                case 4:
                    servo4.write(angle, 40, true);
                    break;
                case 5:
                    servo5.write(angle, 40, true);
                     break;
                }
                if ((angle == 0) || (angle == 180)) {
                    pinState = HIGH;
```

```
                }
                else {
                    pinState = LOW;
                }
                digitalWrite(ledPin, pinState);
            }
        }
    }
```

- **Movimiento de la estructura mediante Python y Arduino**

En el siguiente código Python, junto al código en Arduino, se realiza una cinemática directa, conectándose a Arduino y llevando cada articulación a la posición indicada. En este apartado se realiza control sólo sobre la estructura física. El código en Python sería:

```
# -*- coding: utf-8 -*-
"""
BRAZO ANTROPOMORFICO

CONTROL DE LA ESTRUCTURA FISICA MEDIANTE EL ENVIO DE COMANDOS DESDE PYTHON
"""
from math import *
import random
import numpy as np
import ctypes
import time
import serial

usbport = '/dev/ttyACM1'
s = serial.Serial(usbport, 9600, timeout=1)

print ('CONEXION PYTHON/ARDUINO')
```

```
#### FUNCIONES ARDUINO ####

def move(servo, ang):
    s.write(chr(255))
    s.write(chr(servo))
    s.write(chr(ang))

"""
SE ENVIA A ARDUINO DOS DATOS: EL NÚMERO DEL SERVO A MOVER Y EL ÁNGULO AL QUE DEBE LLEGAR
"""
##INICIALIZACIONES
"""
SE LLEVA A CADA UNA DE LAS ARTICULACIONES A UNA POSICION DE PARTIDA
"""
##ARTICUALCION 1
move(1, 90)
 ## ARTICULACION 2
move(2, 90)
 ## ARTICULACION 3
move(3, 90)
## ARTICULACION 4
move(4, 90)
## ARTICULACION 5
move(5, 180)

#SIMULACION
interrupcion = 0
opcion = 100

while (opcion != interrupcion):
```

```
print ('')
print ('INDIQUE LA ARTICULACION A MOVER')
opcion = input()

if (opcion == 1):
    time.sleep(1)
    print ('')
    print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA ARTICULACION 1')
    angg = input()

    if (0 <= angg <= 180):
        move(1, angg)
        time.sleep(2)

    else:
        print ('')
        print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')

elif (opcion == 2):
    time.sleep(1)
    print ('')
    print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA ARTICULACION 2')
    angg = input()

    if (0 <= angg <= 180):
        move(2, angg)
        time.sleep(2)

    else:
        print ('')
        print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')
```

```
    elif (opcion == 3):
        time.sleep(1)
        print ('')
        print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA ARTICULACION 3')
        angg = input()
        if (0 <= angg <= 180):
            move(3, angg)
            time.sleep(2)

        else:
            print ('')
            print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')

    elif (opcion == 4):
        time.sleep(1)
        print ('')
        print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA ARTICULACION 4')
        angg = input()

        if (0 <= angg <= 180):
            move(4, angg)
            time.sleep(2)

        else:
            print ('')
            print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')

    elif (opcion == 5):
        time.sleep(1)
        print ('')
        print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA ARTICULACION 5')
        print ('LIMITES DE LA PINZA: .0=CERRADA. .40=ABIERTA.')
        angg = input()
```

```
        if (0 <= angg <= 40):
            move(5, (360-(3*angg))/2)
# EQUIVALENCIA DEL ANGULO ESTABLECIDO EN EL CODIGO (0 A 40) Y EL
# FIJADO EN EL SERVOMOTOR (180 A 120)
            time.sleep(2)

        else:
            print ('')
            print ('EL GIRO DEL SERVO ESTA LIMITADO A 40 GRADOS')

    elif (opcion == 9):
        time.sleep(1)
        print ('')
        print ('LLEVANDO A LAS ARTICULACIONES A SUS POSICIONES INICIALES')
        move(1, 90)
        time.sleep(1)
        move(2, 90)
        time.sleep(1)
        move(3, 90)
        time.sleep(1)
        move(4, 90)
        time.sleep(1)
        move(5, 180)
        time.sleep(2)

    elif (opcion == 0):
        print ('')
        print ('FINALIZANDO LA SIMULACION')

    else:
        print ('')
        print ('ERROR AL INDICAR LA ARTICULACION (OPCIONES DE 1 A 5)')
```

```
print ('INTRODUZCA UN 0 PARA SALIR DE LA SIMULACION')
print ('INTRODUZCA UN 9 PARA LLEVAR LAS ARTICULACIONES DEL BRAZO')
print (' A SUS POSICIONES INICIALES')
```

- **Movimiento de la estructura y de un modelo en 3D con piezas puras V-REP.**

En este objetivo se ha conseguido crear un modelo que simula el brazo mediante piezas puras de V-REP y donde se mueve al mismo tiempo el brazo físico. Los códigos utilizados, además del código Arduino utilizado en todos los objetivos, son los códigos en Python utilizados para también para el siguiente apartado/objetivo.

- **Movimiento de la estructura y su representación en V-REP.**

En este caso, se ha conseguido recrear un modelo 3D que representa al modelo físico real. El código en Python realiza un control de cinemática directa y controla el modelo real y su simulación en V-REP al mismo tiempo. El código Python, aparte del código Arduino también utilizado, es:

```
# -*- coding: utf-8 -*-
"""
BRAZO ANTROPOMORFICO. CONTROL SOBRE V-REP Y MODELO FÍSICO DESDE
PYTHON
"""

from math import *
import random
import numpy as np
import ctypes

try:
import vrep

except:
print ('--------------------------------------------------------------')
print ('NO SE PUDO IMPORTAR <<VREP.PY>>. PROBABLEMENTE, <<VREP.PY>> O')
print ('LA BIBLIOTECA <<REMOTE-API>> NO SE PUEDE ENCONTRAR.')
print ('ASEGURESE DE QUE AMBOS ESTÁN EN LA MISMA CARPETA QUE ESTE')
print ('ARCHIVO O AJUSTE ADECUADAMENTE EL ARCHIVO <<VREP.PY>')
```

```
print ('------------------------------------------------------------')
print (")

import time
import serial

usbport = '/dev/ttyACM1'
s = serial.Serial(usbport, 9600, timeout=1)

print ('CONEXION V-REP/PYTHON/ARDUINO')

#### FUNCIONES GENERICAS ####

#### FUNCIONES V-REP ####

def conectarVREP():
        vrep.simxFinish(-1) # CERRAR TODAS LAS CONEXIONES ABIERTAS
        clientID = vrep.simxStart('127.0.0.1',19999,True,True,5000,5)
        # CONECTAR A V-REP

        if (clientID != -1):
            print ('CONECTADO AL SERVIDOR DE API REMOTA')
            vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot_wait)
            vrep.simxSynchronous(clientID,True) # MODO SINCRONO
            return clientID

        else:
            print ('ERROR: PROBLEMA AL CONECTAR CON EL SERVIDOR DE API')
            print ('REMOTA')
            sys.exit(0);

def iniciarSim(clientID):
        vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot)

def pararSim(clientID):
        vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot_wait)

def desconectarVREP(clientID):
        vrep.simxFinish(clientID)
        print ('SESION CON V-REP FINALIZADA')

#### FUNCIONES ARDUINO ####

def move(servo, ang):
        s.write(chr(255))
        s.write(chr(servo))
        s.write(chr(ang))

#### CONECTAR CON VREP ####
```

```
clientID=conectarVREP()


#### INICIALIZACIONES ####

## ARTICULACION 1
ret,joint1_handler = vrep.simxGetObjectHandle(clientID,"Joint1",vrep.simx_opmode_oneshot_wait)
ret,joint1 = vrep.simxGetJointPosition(clientID,joint1_handler,vrep.simx_opmode_streaming)
move(1, 90)

## ARTICULACION 2
ret,joint2_handler = vrep.simxGetObjectHandle(clientID,"Joint2",vrep.simx_opmode_oneshot_wait)
ret,joint2 = vrep.simxGetJointPosition(clientID,joint2_handler,vrep.simx_opmode_streaming)
move(2, 90)

## ARTICULACION 3
ret,joint3_handler = vrep.simxGetObjectHandle(clientID,"Joint3",vrep.simx_opmode_oneshot_wait)
ret,joint3 = vrep.simxGetJointPosition(clientID,joint3_handler,vrep.simx_opmode_streaming)
move(3, 90)

## ARTICULACION 4
ret,joint4_handler = vrep.simxGetObjectHandle(clientID,"Joint4",vrep.simx_opmode_oneshot_wait)
ret,joint4 = vrep.simxGetJointPosition(clientID,joint4_handler,vrep.simx_opmode_streaming)
move(4, 90)

## ARTICULACION 5
ret,joint5_handler = vrep.simxGetObjectHandle(clientID,"Joint5",vrep.simx_opmode_oneshot_wait)
ret,joint5 = vrep.simxGetJointPosition(clientID,joint5_handler,vrep.simx_opmode_streaming)
move(5, 180)


#### SIMULACION ####
iniciarSim(clientID)

interrupcion = 0
opcion = 100

while (opcion != interrupcion):

        print (")
        print ('POSICION EN GRADOS DE LAS ARTICULACIONES:')
        ret,joint1 = vrep.simxGetJointPosition(clientID,joint1_handler,vrep.simx_opmode_buffer)
        print ('1: ' + str(int(joint1*(180/pi))))

        ret,joint2 = vrep.simxGetJointPosition(clientID,joint2_handler,vrep.simx_opmode_buffer)
        print ('2: ' + str(int(joint2*(180/pi))))
        ret,joint3 = vrep.simxGetJointPosition(clientID,joint3_handler,vrep.simx_opmode_buffer)
        print ('3: ' + str(int(joint3*(180/pi))))

        ret,joint4 = vrep.simxGetJointPosition(clientID,joint4_handler,vrep.simx_opmode_buffer)
```

```
print ('4: ' + str(int(joint4*(180/pi))))

ret,joint5 = vrep.simxGetJointPosition(clientID,joint5_handler,vrep.simx_opmode_buffer)
print ('5: ' + str(int(joint5*(180/pi))))


time.sleep(1)
print ('')
print ('INDIQUE LA ARTICULACION A MOVER.')
opcion = input()

if (opcion == 1):
    time.sleep(1)
    print ('')
     print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA')
    print ('ARTICULACION 1')
    angg = input()

    if (0 <= angg <= 180):
      angr = (angg*(pi/180))
      ret =vrep.simxSetJointPosition(clientID,joint1_handler,angr,vrep.simx_opmode_oneshot)
      move(1, angg)
      time.sleep(2)

    else:
      print ('')
      print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')

elif (opcion == 2):
    time.sleep(1)
    print ('')
    print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA')
    print ('ARTICULACION 2')
    angg = input()

    if (0 <= angg <= 180):
      angr = (angg*(pi/180))
      ret =vrep.simxSetJointPosition(clientID,joint2_handler,angr,vrep.simx_opmode_oneshot)
      move(2, angg)
      time.sleep(2)

    else:
      print ('')
      print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')

elif (opcion == 3):
    time.sleep(1)
    print ('')
    print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA')
    print ('ARTICULACION 3')
```

```
            angg = input()


            if (0 <= angg <= 180):
              angr = (angg*(pi/180))
              ret =vrep.simxSetJointPosition(clientID,joint3_handler,angr,vrep.simx_opmode_oneshot)
              move(3, angg)
              time.sleep(2)

            else:
              print ('')
              print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')

        elif (opcion == 4):
            time.sleep(1)
            print ('')
            print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA')
            print ('ARTICULACION 4')
            angg = input()

            if (0 <= angg <= 180):
              angr = (angg*(pi/180))
              ret =vrep.simxSetJointPosition(clientID,joint4_handler,angr,vrep.simx_opmode_oneshot)
              move(4, angg)
              time.sleep(2)

            else:
              print ('')
              print ('EL GIRO DEL SERVO ESTA LIMITADO A 180 GRADOS')

        elif (opcion == 5):
            time.sleep(1)
            print ('')
            print ('POSICION EN GRADOS A LA QUE QUIERE MOVER LA')
            print ('ARTICULACION 5')
            print ('LIMITES DE LA PINZA: .0=CERRADA. .40=ABIERTA.')
            angg = input()

            if (0 <= angg <= 40):
              angr = (angg*(pi/180))
              ret =vrep.simxSetJointPosition(clientID,joint5_handler,angr,vrep.simx_opmode_oneshot)
              move(5, (360-(3*angg))/2)
              # EQUIVALENCIA DEL ANGULO ESTABLECIDO EN V-REP CON EL
              # ENVIADO
              time.sleep(2)

            else:
              print ('')
              print ('EL GIRO DEL SERVO ESTA LIMITADO A 40 GRADOS')
```

```
        elif (opcion == 9):
            time.sleep(1)
            print ('')
            print ('LLEVANDO A LAS ARTICULACIONES A SUS POSICIONES')
            print ('INICIALES')
            ret = vrep.simxSetJointPosition(clientID,joint1_handler,pi/2,vrep.simx_opmode_oneshot)
            move(1, 90)
            time.sleep(1)
            ret = vrep.simxSetJointPosition(clientID,joint2_handler,pi/2,vrep.simx_opmode_oneshot)
            move(2, 90)
            time.sleep(1)
            ret = vrep.simxSetJointPosition(clientID,joint3_handler,pi/2,vrep.simx_opmode_oneshot)
            move(3, 90)
            time.sleep(1)
            ret = vrep.simxSetJointPosition(clientID,joint4_handler,pi/2,vrep.simx_opmode_oneshot)
            move(4, 90)
            time.sleep(1)
            ret = vrep.simxSetJointPosition(clientID,joint5_handler,0,vrep.simx_opmode_oneshot)
            move(5, 180)
            time.sleep(2)

        elif (opcion == 0):
            print ('')
            print ('FINALIZANDO LA SIMULACION')

        else:
            print ('')
            print ('ERROR AL INDICAR LA ARTICULACION (OPCIONES DE 1 A 5)')
            print ('INTRODUZCA UN 0 PARA SALIR DE LA SIMULACION')
            print ('INTRODUZCA UN 9 PARA LLEVAR LAS ARTICULACIONES DEL')
            print ('BRAZO A SUS POSICIONES INICIALES')

time.sleep(2)
pararSim(clientID)

time.sleep(2)
desconectarVREP(clientID)
```

En este código de Python se comienza por establecer una conexión tanto con Arduino como con el V-REP mediante la API Remota mediante diversas funciones. Posteriormente, se da paso a las inicializaciones de V-REP y de las articulaciones en Arduino.

Por otro lado, dentro del primer bucle se nos mostrará por pantalla la posición de las articulaciones en grados en cada momento para después pedirnos introducir la articulación

que deseamos mover y el ángulo al que queremos llevar la articulación.

Dado el uso de servomotores que solamente se mueven entre los 0º y 180º existirá un límite para el usuario, no pudiendo poner un valor por fuera de este rango. En el caso de la pinza, al tener un recorrido menor, se ha establecido 0º como pinza cerrada y 40º como pinza abierta, pudiendo variar solamente dentro de estos límites. Siempre se introducen los valores en grados, dado que el código ya lleva una conversión de radianes a grados.

Finalmente, existen dos casos más por el que el usuario puede optar. Si introduce un "9" cuando se le pide una articulación se llevarán todas las articulaciones a sus posiciones iniciales. En cambio, si se introduce un "0" se parará la simulación.

Cabe destacar, de las funciones de la API Remota más importantes en este código [15]:

- vrep.simxGetJointPosition: Utilizada para obtener la posición de una articulación en V-REP .
- vrep.simxSetJointPosition: Utilizada para dar la posición a una articulación en V-REP.

- **Movimiento del modelo real y su representación con "sliders" desde V-REP.**

En este caso se utilizan tres códigos distintos. Se recogen los datos y posiciones de cada articulación de V-REP mediante un código Python. A su vez, en V-REP existe un código implementado, en LUA, que se ocupa de dar posiciones a las articulaciones mediante "sliders" o botones deslizantes. De esta manera, en este objetivo logrado sería desde V-REP donde se envían las posiciones de las articulaciones y Python funcionaría como puente entre V-REP y Arduino. Hay que tener en cuenta, que en este caso en Arduino se utiliza la librería "Servo.h", dado que el control de velocidad lo realiza el usuario mediante el movimiento de los *sliders*.

El código en Arduino en este caso sería:

```
#include <Servo.h>

// SE INCLUYE LA LIBRERIA <Servo.h> PARA DECLARAR LOS SERVOS
Servo servo1;
```

```
Servo servo2;
Servo servo3;
Servo servo4;
Servo servo5;

// MINIMO Y MAXIMO PULSO DE UN SERVO PARA IR DE 0º A 180º
int minPulse = 600;
int maxPulse = 2400;
int userInput[2];
int incomingByte;

int servoNumber;
int angle;
int i;
int ledPin = 13;
int pinState = LOW;

void setup() {
    Serial.begin(9600);

// SE ASIGNA A CADA SERVO DECLARADO UN PIN DEL ARDUINO
    servo1.attach(3, minPulse, maxPulse);
    servo2.attach(5, minPulse, maxPulse);
    servo3.attach(6, minPulse, maxPulse);
    servo4.attach(9, minPulse, maxPulse);
    servo5.attach(10, minPulse, maxPulse);
    pinMode(ledPin, OUTPUT);

// SE ENVIA A LOS SERVOS A SUS POSICIONES INICIALES
    servo1.write(90);
    servo2.write(90);
    servo3.write(90);
    servo4.write(90);
```

```
        servo5.write(180);
    }

    void loop() {
        if (Serial.available() > 2) {
    // SE ESPERA RECIBIR MAS DE DOS DATOS
            incomingByte = Serial.read();
    // EL PRIMER DATO RECIBIDO ES PARA DETERMINAR QUE ESTAMOS ANTE
    // LA RECEPCIÓN DE UN MENSAJE QUE LLEVA ASOCIADO 2 DATOS
    // NUMERO DE SERVO Y ANGULO
            if (incomingByte == 0) {
                for (i=0; i<2; i++) {
                    userInput[i] = Serial.read();
                    }
                    servoNumber = userInput[0];
                    angle = userInput[1];

                    switch (servoNumber) {
                    case 1:
                        servo1.write(angle);
                        break;
                    case 2:
                        servo2.write(angle);
                        break;
                    case 3:
                        servo3.write(angle);
                        break;
                    case 4:
                        servo4.write(angle);
                        break;
                    case 5:
                        servo5.write(angle);
                         break;
```

```
                }
                if ((angle == 0) || (angle == 180)) {
                    pinState = HIGH;
                }
                else {
                    pinState = LOW;
                }
                digitalWrite(ledPin, pinState);
            }
        }
    }
```

El código en Python utilizado es:

```
# -*- coding: utf-8 -*-
"""
BRAZO ANTROPOMORFICO CONTROLADO DESDE V-REP. PYTHON ACTUA
COMO PUENTE ENTRE V-REP Y ARDUINO
"""
from math import *
import random
import numpy as np
import ctypes

try:

        import vrep

except:

        print ('--------------------------------------------------------------')
        print ('NO SE PUDO IMPORTAR <<VREP.PY>>. PROBABLEMENTE, ')
        print ('<<VREP.PY>> O LA 'BIBLIOTECA <<REMOTE-API>> NO SE PUEDE')
        print ('ENCONTRAR. ASEGURESE DE QUE 'AMBOS ESTÁN EN LA MISMA')
        print ('CARPETA QUE ESTE ARCHIVO O AJUSTE 'ADECUADAMENTE EL')
        print ('ARCHIVO <<VREP.PY>')
        print ('--------------------------------------------------------------')
        print ('')

import time

import serial

usbport = '/dev/ttyUSB0'
```

```
s = serial.Serial(usbport, 9600, timeout=1)

print ('CONEXION V-REP/PYTHON/ARDUINO')

#### FUNCIONES GENERICAS ####

#### FUNCIONES V-REP ####

def conectarVREP():
        vrep.simxFinish(-1) # CERRAR TODAS LAS CONEXIONES ABIERTAS
        clientID = vrep.simxStart('127.0.0.1',19999,True,True,5000,5)
        # CONECTAR A V-REP

        if (clientID != -1):
            print ('CONECTADO AL SERVIDOR DE API REMOTA')
            vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot_wait)
            vrep.simxSynchronous(clientID,True) # MODO SINCRONO
            return clientID

        else:
            print ('ERROR: PROBLEMA AL CONECTAR CON EL SERVIDOR DE API
REMOTA')
            sys.exit(0);

def iniciarSim(clientID):
        vrep.simxStartSimulation(clientID,vrep.simx_opmode_oneshot)

def pararSim(clientID):
        vrep.simxStopSimulation(clientID,vrep.simx_opmode_oneshot_wait)
def desconectarVREP(clientID):
        vrep.simxFinish(clientID)
        print ('SESION CON V-REP FINALIZADA')
#### FUNCIONES ARDUINO ####

def move(servo, ang):
        s.write(chr(0))
        s.write(chr(servo))
        s.write(chr(ang))

#### CONECTAR CON VREP ####

clientID = conectarVREP()

#### INICIALIZACIONES ####

## ARTICULACION 1

ret,joint1_handler = vrep.simxGetObjectHandle(clientID,"Joint1",vrep.simx_opmode_oneshot_wait)
ret,joint1 = vrep.simxGetJointPosition(clientID,joint1_handler,vrep.simx_opmode_streaming)
move(1, 90)
```

```
## ARTICULACION 2

ret,joint2_handler = vrep.simxGetObjectHandle(clientID,"Joint2",vrep.simx_opmode_oneshot_wait)
ret,joint2 = vrep.simxGetJointPosition(clientID,joint2_handler,vrep.simx_opmode_streaming)
move(2, 90)

## ARTICULACION 3

ret,joint3_handler = vrep.simxGetObjectHandle(clientID,"Joint3",vrep.simx_opmode_oneshot_wait)
ret,joint3 = vrep.simxGetJointPosition(clientID,joint3_handler,vrep.simx_opmode_streaming)
move(3, 90)

## ARTICULACION 4

ret,joint4_handler = vrep.simxGetObjectHandle(clientID,"Joint4",vrep.simx_opmode_oneshot_wait)
ret,joint4 = vrep.simxGetJointPosition(clientID,joint4_handler,vrep.simx_opmode_streaming)
move(4, 90)

## ARTICULACION 5

ret,joint5_handler = vrep.simxGetObjectHandle(clientID,"Joint5",vrep.simx_opmode_oneshot_wait)
ret,joint5 = vrep.simxGetJointPosition(clientID,joint5_handler,vrep.simx_opmode_streaming)
move(5, 180)

#### SIMULACION ####

iniciarSim(clientID)
while (True):

        print (")
        print ('POSICION EN GRADOS DE LAS ARTICULACIONES:')

        ret,joint1 = vrep.simxGetJointPosition(clientID,joint1_handler,vrep.simx_opmode_buffer)
        print ('1: ' + str(int(joint1*(180/pi))))
        move(1, int(joint1*(180/pi)))


        ret,joint2 = vrep.simxGetJointPosition(clientID,joint2_handler,vrep.simx_opmode_buffer)
        print ('2: ' + str(int(joint2*(180/pi))))
        move(2, int(joint2*(180/pi)))


        ret,joint3 = vrep.simxGetJointPosition(clientID,joint3_handler,vrep.simx_opmode_buffer)
        print ('3: ' + str(int(joint3*(180/pi))))
        move(3, int(joint3*(180/pi)))


        ret,joint4 = vrep.simxGetJointPosition(clientID,joint4_handler,vrep.simx_opmode_buffer)
        print ('4: ' + str(int(joint4*(180/pi))))
        move(4, int(joint4*(180/pi)))
```

```
        ret,joint5 = vrep.simxGetJointPosition(clientID,joint5_handler,vrep.simx_opmode_buffer)
        print ('5: ' + str(int(joint5*(180/pi))))
        angle = ((360-(3*int(joint5*(180/pi))))/2)
        move(5, angle)
        # EQUIVALENCIA DEL ANGULO ESTABLECIDO EN V-REP CON EL
        # ENVIADO


time.sleep(2)
pararSim(clientID)

time.sleep(2)
desconectarVREP(clientID)
```

Tras las inicializaciones en este código en Python podemos observar como dentro del bucle se adquieren cada una de las posiciones de cada articulación en V-REP para enviarlas a Arduino.

El código en LUA utilizado para dar el ángulo a cada posición mediante los “sliders” desde V-REP es:

```
-- DO NOT WRITE CODE OUTSIDE OF THE if-then-end SECTIONS BELOW!!
-- (unless the code is a function definition)

if (sim_call_type==sim_childscriptcall_initialization) then


  if (simGetScriptExecutionCount()==0) then
      --Codigo inicializacion
      ctrlID=simGetUIHandle("UI")
      sliderID={4,6,8,10,12}
   end

   --Codigo inicializacion
```

```
    qID={}
    for i=1,5,1 do
    qID[i]=simGetObjectHandle('Joint' .. i)
    end

end

if (sim_call_type==sim_childscriptcall_actuation) then

    simHandleChildScripts(sim_handle_all_except_explicit)
        --Codigo main
    print('######################')
    print('SLIDERS')
    for i=1,#sliderID,1 do
        slider_i = simGetUISlider(ctrlID,sliderID[i])
       -- print('slider' .. i .. '=' .. slider_i) Mostraremos solamente un angulo entre 0 y 180 grados
        slider_iconv = (slider_i*0.18)
        print ('slider' .. i .. '=' .. slider_iconv)

    end

  print('Posiciones de las articulaciones')

    for i=1,#qID,1 do
        joint_i=simGetJointPosition(qID[i])
        joint_iconv=((joint_i/(2*math.pi))*360)
        print('La posicion de la joint' .. i .. ' es ' .. joint_iconv)
    end
    for i=1,5,1 do
        slider_i = simGetUISlider(ctrlID,sliderID[1])
        slider_iconv = (slider_i*0.18)
        slider_iconvrad = ((slider_iconv/180)*(math.pi))
       for i=1,4,1 do
```

```
        result=simSetJointPosition(qID[1],slider_iconvrad)
    end
end
for i=1,#sliderID,1 do
    slider_i = simGetUISlider(ctrlID,sliderID[2])
    slider_iconv = (slider_i*0.18)
    slider_iconvrad = ((slider_iconv/180)*(math.pi))
  for i=1,#qID,1 do
        result=simSetJointPosition(qID[2],slider_iconvrad)
    end
end
for i=1,#sliderID,1 do
    slider_i = simGetUISlider(ctrlID,sliderID[3])
    slider_iconv = (slider_i*0.18)
    slider_iconvrad = ((slider_iconv/180)*(math.pi))
  for i=1,#qID,1 do
        result=simSetJointPosition(qID[3],slider_iconvrad)
    end
end




for i=1,#sliderID,1 do
    slider_i = simGetUISlider(ctrlID,sliderID[4])
    slider_iconv = (slider_i*0.18)
    slider_iconvrad = ((slider_iconv/180)*(math.pi))
  for i=1,#qID,1 do
        result=simSetJointPosition(qID[4],slider_iconvrad)
    end
end

for i=1,#sliderID,1 do
```

```
        slider_i = simGetUISlider(ctrlID,sliderID[5])
        slider_iconv = (slider_i*0.04)
        slider_iconvrad = ((slider_iconv/180)*(math.pi))
       for i=1,#qID,1 do
           result=simSetJointPosition(qID[5],slider_iconvrad)
       end
    end
end
if (sim_call_type==sim_childscriptcall_sensing) then

        -- Put your main SENSING code here

end

if (sim_call_type==sim_childscriptcall_cleanup) then

        -- Put some restoration code here

end
```

En V-REP se crean los "sliders" de manera visual, para posteriormente programar su funcionamiento en el código. Como se puede ver, en las inicializaciones del código LUA se utilizan las funciones "simGetUIHandle" y "simGetObjectHandle" para obtener los datos de referencia al "slider" u objeto (en este caso "joints") respectivamente [14].

Dado esto, se procede al código main. En este se comenzará por mostrar por consola en todo momento las posiciones de las diferentes articulaciones. Resuelto este punto, se realizan diferentes "for" para obtener de cada uno los "sliders" creados la posición en la que se encuentra el botón deslizante. Tras conocer esta posición se hace una conversión numérica ya que los valores de "sliders" van de 0 a 1000 y a nosotros nos interesa que sea de 0 a 180

(menos en la pinza que va de 0 a 40). Al realizar la conversión se adjudica, utilizando la función simSetJointPosition, cada una de esas posiciones de los botones deslizantes a cada una de las articulaciones del modelo.