Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica.

Diseño y Control de un Robot Manipulador

Autor: Jesús Tormo Barbero
Tutor: José Ángel Acosta Rodríguez

Proyecto Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica.

Diseño y Control de un Robot Manipulador

Autor:
Jesús Tormo Barbero

Tutor:
José Ángel Acosta Rodríguez
Profesor titular

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Grado: Diseño y Control de un Robot Manipulador

Autor: Jesús Tormo Barbero

Tutor: José Ángel Acosta Rodríguez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A quienes me empujaron a comenzar el camino; al que conmigo lo reccorió y al que apareció en él.

A quien seguirá en el que está por venir.

A todos, ¡Gracias!

Agradecimientos

Quiero aprovechar la ocasión para mostrar mi gratitud a todo aquel que ha estado conmigo en todo este tiempo y dándome cada uno una pizca de ellos mismos, ya que al final, acabamos siendo un poco de cada persona que comparte el tiempo con nosotros.

Así, quiero agradecer a mi tutor Jose Ángel Acosta todo el apoyo que me ha dado, la implicación que ha tenido desde primera hora conmigo y con el trabajo, por dejar que volara mi imaginación y permitir desarrollar las ideas que se me ocurrían y ayudarme a que se lograran. Muchas gracias por todo, hay pocos como tu.

No se puede dejar atrás a Carlos Rodríguez, que se ha implicado en mi proyecto como si fuera suyo, ofreciéndome multitud de consejos e ideas que han permitido que el proyecto concluyera como es debido.

Por supuesto, a las personas que más quiero, a mi padre y a mi madre, al día y a la noche, al que es todo razón y a la que es todo corazón, aunque nadie tiene un corazón más grande que mi padre, ni más sabiduría que mi madre. Gracias a mi padre, por cada consejo, por cada palabra de apoyo y el quitarme peso de presión de encima, por ayudarme con el proyecto en todo momento, por el apoyo proporcionado y por su paciencia/exigencia. Gracias a mi madre, sin tus ánimos y tu risa no habría conseguido nada, por permitirme tenerle todo patas arriba, por los abrazos y las cruces. Gracias a los dos por permitirme vivir esta experiencia. Soy como soy gracias a vosotros, nunca podré agradeceros todo lo que habéis hecho por mi.

A mi hermana Cristi, por su autenticidad y su fuerza, por ser siempre un pilar en mi vida, por tener una persona en quien confiar y a la que acudir cuando todo parece ir mal, por todas las cosas que hace una hermana grande a lo largo de su vida por su hermanito pequeño. No sabes cuanto te quiero.

A toda mi familia, en especial a mi abuelita, que me ha acogido durante todo el proyecto en su casa dándome la merienda. A mis tios y primos, por darme todo su apoyo y motivación, no hace falta que os diga lo importantes que sois todos para mi y cuanto os quiero. Gracias.

No puede faltar mis palabras hacia mi familia elegida, cada una de las personas que me llevan aguantando desde que puse un pie en la calle, con los que he crecido y avanzado, con los que he compartido tantos momentos buenos y no tan buenos, aquellos que se que puedo contar en cualquier momento… Hay personas que dicen poder contar con una mano a sus amigos de verdad, yo tengo la suerte de no compartir esa misma opinión. ¡¡Va por vosotros, Escort!!

Y por supuesto a mi banda, por proporcionarme una cama día si y día también, por haber vivido esta aventura juntos, y acabar consiguiendo el objetivo sudando sangre, pero siempre juntos y apoyándonos, y lo más importante, creando un lazo entre nosotros que no se perderá. Ha merecido la pena estos años en la escuela solo por haberos conocido.

Y sí, tampoco me olvido de ti, gracias por todo este tiempo guapa, nunca cambies ni dejes de ser tu misma. Te quiero enana.

Y para terminar, gracias a todos y cada uno de las personas con las que he convivido día a día en la escuela, en especial, a aquellos profesores que de verdad han disfrutado enseñando, vosotros marcáis la diferencia.

No tengo forma de agradeceros todo lo que habéis hecho y hacéis por mi, espero poder hacerlo algún día. Os quiero.

Jesús Tormo Barbero

Sevilla, 2018

Resumen

La robótica cobra cada vez más fuerza en nuestros tiempos. Cada día que pasa, consigue tener un papel más importante, afectando de manera más directa a las vidas de las personas. Las barreras existentes entre la ficción y la realidad se acaban rompiendo y los escritores de ciencia ficción acaban por ser profetas que eran capaces de ver hasta donde va a llegar la humanidad.

Este cambio necesita de un proceso, el cuál conlleva una gran componente experimental. Surge la necesidad de probar los conceptos teóricos y así poder obtener los resultados que guiarán el camino hasta conseguir el objetivo final. Surge, por tanto, la necesidad de prototipar.

En este Trabajo de Fin de Grado se ha realizado el prototitado para probar nuevos controladores de brazos robóticos. Para ello se ha realizado un estudio previo de los componentes, adquisición de nuevos elementos, así como el diseño, fabricación y programación de los dos brazos robóticos con visión artificial, que se usarán para la experimentación de los nuevos controladores.

Abstract

Robotics is gaining strength in our times. Each day that passes, it has a more important role, affecting more directly the lives of people. The barriers between fiction and reality are breaking down, and science fiction writers end up being prophets who were able to see how far humanity is going to go.

This change requires a process, which involves a large experimental component. The need arises to prove the theoretical concepts and thus be able to obtain the results that will guide the way until achieving the final objective. Therefore, there is a need to prototype.

In this End-of-Degree Work, the prototype has been made to test new robotic arms controllers. For this, a preliminary study of the components has been carried out, acquisition of new elements, as well as the design, manufacture and programming of the two robotic arms with artificial vision, which will be used for the experimentation of the new controllers.

... -translation by google-

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ECUACIONES

1 INTRODUCCIÓN.

"Si no puedes volar entonces corre, si no puedes correr entonces camina, si no puedes caminar entonces arrástrate, pero sea lo que sea que hagas, sigue moviéndote hacia adelante."

- Martin Luthr King, Jr. -

A lo largo de este primer capitulo se van a desarrollar diferentes aspectos que se consideran importantes para la comprensión del trabajo que nos ocupa.

Se considera de vital importancia justificar la realización de este proyecto ya que no solo aborda aspectos teóricos sobre los robots, sino que también dispone de un fuerte componente de desarrollo práctico, llevando a cabo el prototipado de dos brazos robóticos.

1.1 Justificación del Proyecto.

Cuando se intenta llevar una idea a la realidad, surgen aspectos que no habían sido tenidos en cuenta, de ahí la importancia de que existan prototipos en los cuales se comprueben los contratiempos que puedan surgir, así como solucionar los problemas en versiones de pruebas que ahorran dinero y esfuerzo para la versión final.

En este proyecto se ha realizado un prototipo funcional de un sistema en el cuál dos brazos robóticos interactuarán entre ellos. Hay que destacar que se pretende probar el funcionamiento de un nuevo tipo de control para brazos robóticos industriales y compliant.

Este nuevo tipo de robot se caracteriza por tener uniones entre los grados de libertad que son flexibles y disponen de amortiguación, posibilitando una amplia cantidad de nuevas funcionalidades y horizontes en el mundo de la robótica.

1.2 Objetivos:

En un principio, el objetivo de este Trabajo Fin de Grado consistía en diseñar un sistema robótico onteniendo el máximo partido posible al Kit Bioloid Premium de la empresa ROBOTIS.

Tras realizar un trabajo de investigación sobre los componentes de kit (servomotores, controlador y piezas), se llega a la conclusión de que los actuadores que ofrece el kit son capaces de ofrecer una funcionalidad muy amplia, pero quedan limitados debido al resto de componentes que los acompañan. Llegados a esta conclusión, se decide diseñar y fabricar nuevas piezas que complementan al kit, así como un nuevo controlador más potente que evita los cuellos de botella que tiene el del kit.

Tras la adquisición de los nuevos elementos que compondrán el proyecto y conscientes del potencial y el rendimiento que se le podía sacar a los actuadores, los objetivos se amplían, adoptando unas metas un poco más ambiciosas.

1.2.1 Objetivo General.

El objetivo principal del Proyecto es realizar un Sistema robótico de pruebas con dos brazos manipuladores y realimentación visual con el fin de experimentar y probar los nuevos controles de robots, en particular, los brazos flexibles.

1.2.2 Objetivos Específicos.

- Diseño, construcción y programación de un brazo robótico industrial de seis grados de libertad que sirva para probar el sistema de control que, posteriormente, se empleará en el robot compliant más complejo.
- Diseño y construcción de un brazo robótico compliant con articulaciones flexibles y sirva como banco de pruebas de controladores de este tipo.
- Realización de un sistema de vision artificial, capaz de complementar las medidas necesarias para el controlador que se deasea implementar en el brazo robótico compliant.

1.3 Estructura de la memoria:

El documento que nos ocupa está organizado de forma secuencial, abordando los apartados de la misma manera que fueron surgiendo a medida que se desarrollaba el proyecto. Seguirá el siguiente orden:

- Hardware → Se presentarán los elementos usados durante el proyecto, como son los controladores, los actuadores y herramientas de fabricación de piezas.
- Sistema → Se divide en tres subapartados en los cuales se explicarán los conceptos, duseño y fabricación de cada uno de los tres objetivos específicos.
- Control y Simulación → Definición de controles usados para los sistemas robóticos, así como las simulaciones.
- Experimentación y Análisis de Resultados → Se expondrán como se han preparado los experimentos realizados, los pseudocódigos implementados y los resultados obtenidos.
- Conclusiones → Se valorará el trabajo realizado una vez concluido el proyecto.
- Anexos → Se podrá encontrar en este apartado toda la información considerada de interés, así como los códigos implementados al completo.

-

2 HARDWARE.

"Nunca te fíes de un ordenador que no puedas tirar por la ventana"

- Steve Wozniak -

A lo largo de este capítulo se realizará una introducción del hardware usado en el proyecto. Se comenzará realizando una definición de los componentes iniciales ofrecidos por el kit Bioloid. Tras ello, se expondrán el resto de los componentes que se han decidido usar para realizar el trabajo, así como una explicación de por qué han sido los elegidos.

2.1 Introducción.

El proyecto comienza con un estudio previo del kit y la experimentación con el mismo, se llega a la conclusión de que ciertos componentes no cumplen los requisitos necesarios para la realización del proyecto, por lo que es necesario buscar soluciones que solventen las carencias de los mismos, y por consiguiente, su reemplazo.

El principal contratiempo que se tuvo fue el controlador CM-530, ya que no permite realizar una programación compleja, y a la hora de realizar la comunicación con la RaspBerry resultaba ser muy poco eficiente, ya que se debería hacer a través de los puertos auxiliares. Para resolver el problema se buscó un sustituto, y el elegido fue el Arbotix-M de Trossen Robotis, el cual se analizará más tarde.

El siguiente problema fue en el diseño de los brazos de robots. Para el caso del robot industrial de 6 grados de libertad, las piezas de las que se disponían resultaban ser limitadas, ya que, a pesar de ser muy versátiles, no se podía diseñar con ellas un prototipo funcional, debido a que las articulaciones inferiores del robot soportaban una gran cantidad de peso y un solo motor no era capaz de mover correctamente el conjunto.

Por tanto, se decidió poner dos motores en paralelo en las articulaciones que necesitaban realizar un mayor esfuerzo, consiguiendo un movimiento más fluido y eficiente, pero para hacerlo se necesitaban piezas diferentes a las ofrecidas en el kit.

Para el robot "Compliant" resultaba ser un problema mayor, ya que decidir hacer este tipo de brazo robótico implica tener articulaciones flexibles entre los motores.

Con el déficit de piezas, surge la necesidad de diseñarlas específicamente cada una de ellas. Para el diseño se utilizó el software de diseño Catia. Una vez creadass las estructuras necesarias para cada uno de los robots, se realizó su impresión 3D.

2.2 Kit Bioloid Premium.

El trabajo de investigación previo ha sido de vital importancia para el desarroyo del proyecto. Fue una labor que llevó bastante tiempo, y el desencadenante del replanteamiento de los objetivos iniciales. Se proseguirá exponiendo el kit y sus componentes.

Figura 2-1. Kit Bioloid Premium

¿Qué es el Kit Bioloid Premium?

"Es un sistema flexible, escalable y reconfigurable con el que se aprende lo básico de estructuras y principios de las articulaciones de robots y ampliar su aplicación a la ingeniería creativa, cinemática inversa y la cinética. También es para los aficionados que disfrutan del diseño y construcción de robots a medida como hobby, prototipos e investigación. Apto para aprender en bachillerato, FP y investigación en universidades. Es el kit de robot más utilizado para participar en competiciones de robots humanoides."

El kit está compuesto por:[1]

- 1 x controladora CM-530 (ARM Cortex M3 / 32bit)
- 18 x servos DYNAMIXEL AX-12A
- 2 x sensores objetos IR
- 1 x sensor distancia precisón Sharp
- 1 x sensor gyro
- 1 x control remoto RC-100A
- 1 x módulos inalámbricos Zizbee ZIG-110 (GRATIS)
- 1 x alimentador de potencia SMPS
- 1 x batería recargable LiPo
- 1 x cargador batería LiPo
- 1 x cable USB - mini USB
- 1 x Cabeza y cubierta de cuerpo
- 1 x Frame Set (estructuras) de plastico
- CD con software RoboPlus
- Destornillador
- Tensores de cable
- Libro QuickStart

[1] El Anexo I se incluye un listado completo de los componentes del Kit.

2.2.1 Controlador CM 530.

El controlador del que dispone el kit se denomina CM-530 (Figura 2-2). Tiene un procesador ARM Cortex STM32F103RE (32 bits). Se conecta al PC por medio de un cable mini USB. Se puede trabajar entre los 6,5 V a 15 V, aunque se aconseja mantener un voltaje constante de 11,1 V, soportando una corriente máxima de 10 A (fusible). Además, dispone de los siguientes puertos auxiliares y sensores:

- 6 puertos auxiliares de 5 pines compatibles con OLLO para la expansión con sensores y dispositivos de terceros: DMS (sensor de distancia Sharp), sensor de contacto OLLO, sensor de IR, etc.
- El puerto para el receptor ZigBee o Bluetooth.
- 5 puertos de bus TTL de Bioloid (3 pin) para controlar en bus serie Daisy-Chain un número grande de actuadores de la serie AX o de los nuevos actuadores de la serie MX de 3 pines, TTL (encoder magnético).
- LEDs informativos del estado del microcontrolador.
- Detector de sonido (micrófono).
- Sensor de temperatura.
- Sensor de tensión.

Figura 2-2. CM-530

El CM-530 es compatible con los actuadores Dynamixel: AX-12W, AX-12A, AX-18A, MX-28T, MX-64T, MX-106T, además de diferentes sensores como un giróscopo, DMS (distancia precisa), tacto e infrarrojos.

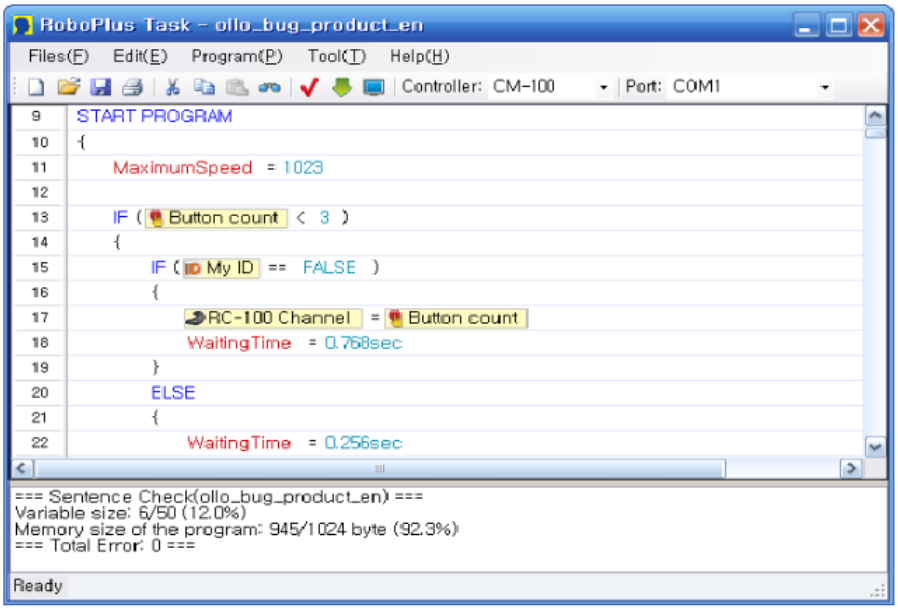
El entorno de programación y control está diseñado especialmente para este controlador, siendo un entorno muy sencillo, pero poco versátil, ya que está pensado para inicializarse en el mundo de la robótica. Se trata del software Suite de programas RoboPlus (descargable gratuitamente).

I. Software de Programacion: RoboPlus.

Este software está diseñado por la empresa ROBOTIS para la programación de sus controladores. Se distinguen cuatro tipos de programación diferentes:[2]

[2] Para mas información sobre RoboPlus, visitar la web: http://www.robotis.us/roboplus1/

- **RoboPlus Task:** Código Fuente en el cuál se le asignan las tareas que deberá hacer el robot. Es un software basado en la programacion C/C++ (Figura 2-3).



- Figura 2-3. RoboPlus Task

- **RoboPlus Manager:** se usa para manejar dispositivos usados por un robot. Las principales funciones de este programa son las siguientes:
  - Administra el firmware del controlador. (Actualización y restauración)
  - Inspecciona el estado del controlador y los dispositivos periféricos. (Prueba)
  - Establece los modos requeridos. (Configuraciones)

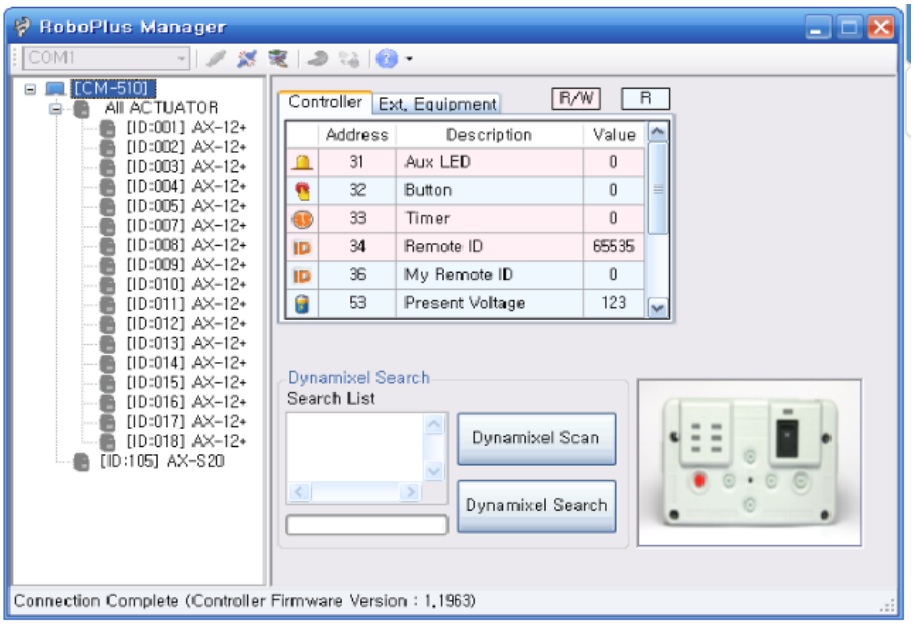


Figura 2-4. RoboPlusManager

- **RoboPlus Motion:** es una herramienta de programación animada que se utiliza para crear movimientos de robot (coreografías) mediante la edición de la velocidad y la posición de Dynamixel.

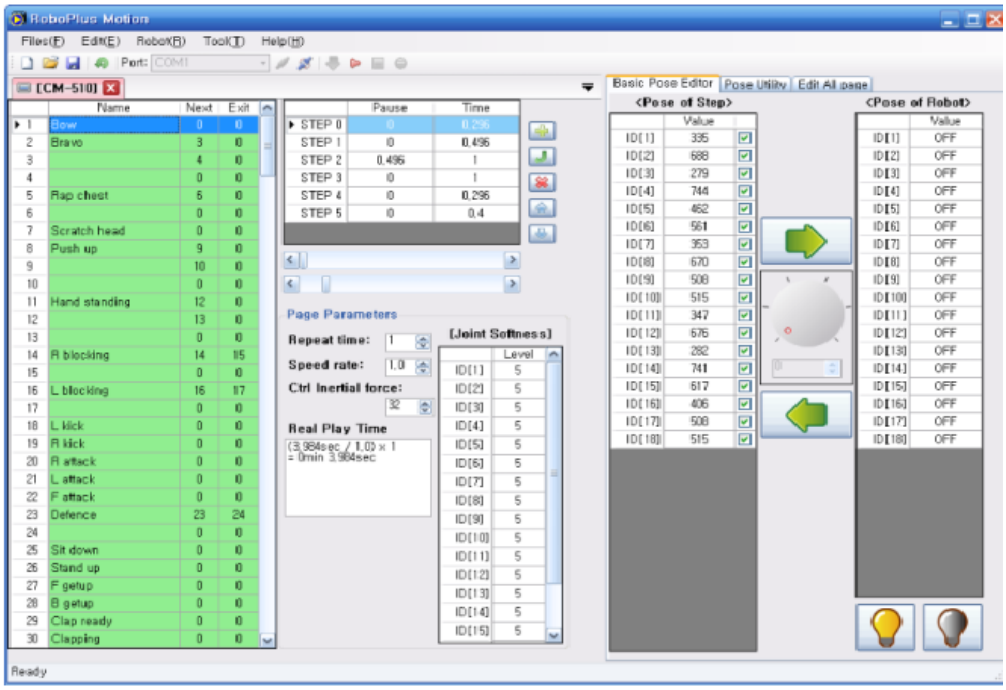


Figura 2-5. RoboPlus Motion

Como se puede comprobar, estos tres módulos abarcan lo que sería la programación completa de un controlador de robot, facilitando su trabajo gracias a una interfaz intuitiva y fácil de programar. Pero también tiene sus inconvenientes; limita mucho la programación propiamente dicha, impidiendo la posibilidad de realizar un programa complejo que nos permita adentrarnos en las entrañas de los sensores, actuadores y comunicación con otros hardware.

2.2.2 Servo Motores AX-12 A (Dynamixel).

El kit contiene 18 servomotores Dynamixel AX-12A. Estos actuadores ofrecen unas prestaciones muy potentes, siendo totalmente programables y proporcionando una gran variedad de datos sobre si mismos. Algunas de las principales características de estos motores son:[3]

- Tienen un ID que los identifica.
- Son controlados por protocolos de comunicación digital de paquetes en la red serie. Soportan TTL y el estándar RS-485 (dependiendo del modelo).
- Están retroalimentados: entre otras se puede leer la posición actual del motor, la velocidad, la temperatura interna, el torque o la tensión de alimentación.
- Disponen de una Tabla de control.
- Función de alarma: cuando la temperatura interna, torque (carga), tensión de alimentación, etc. salen de unos márgenes proporciona retroalimentación sobre la situación y se puede encender un led de alarma o desconectar el actuador.
- Alta eficiencia debido a la baja corriente de consumo.

Figura 2-6. DYNAMIXEL AX-12A

Cada actuador Dynamixel tiene un microcontrolador que se encarga de fijar los parámetros que definen el comportamiento del motor. Esto lo hace gracias a la anteriormente nombrada Tabla de Control, en la cual vamos a profundizar un poco más.

Esta tabla contiene datos sobre el estado actual y la operación que está realizando el servomotor. En función de cómo o qué queramos hacer con el AX-12A, se deberá leer o modificar alguno de los valores que estén en la tabla, que se divide en dos partes:

- EEPROM: Se mantienen los valores una vez establecidos, permaneciendo invariables incluso cuando el motor está desconectado.
- RAM: Los valores son reestablecidos a su valor por defecto cada vez que se reinicia.

Ambas memorias disponen de datos que son de lectura, o lectura y escritura, que son usados para realizar el control del motor. De igual forma, disponen de una dirección que representa la ubicación de los datos. Para leer o escribir en los registros, se necesitará hacer uso de estas direcciones.

Todos los datos están guardados en registros de 8 bits, lo cual permite, en la mayoría de los casos, mantener la información necesaria, pero existen datos como son la posición, velocidad, o torque, que necesitan de dos registros para poder guardar la información al completo.

La forma de usar dos bits en vez de solo uno es nombrando las direcciones con una L para el primer bit (Low → Bajo) y H para el segundo (High → Alto). A continuación, se muestra la tabla al completo para poder obtener más detalles de ella.

[3] Para más información sobre los servomotores AX-12A, mirar el datasheet añadido en la memoria.

❖ **ÁREA EEPROM:**

Address	Item	Access	Initial Value
0(0X00)	Model Number(L)	RD	12(0x0C)
1(0X01)	Model Number(H)	RD	0(0x00)
2(0X02)	Version of Firmware	RD	?
3(0X03)	ID	RD,WR	1(0x01)
4(0X04)	Baud Rate	RD,WR	1(0x01)
5(0X05)	Return Delay Time	RD,WR	250(0xFA)
6(0X06)	CW Angle Limit(L)	RD,WR	0(0x00)
7(0X07)	CW Angle Limit(H)	RD,WR	0(0x00)
8(0X08)	CCW Angle Limit(L)	RD,WR	255(0xFF)
9(0X09)	CCW Angle Limit(H)	RD,WR	3(0x03)
10(0x0A)	(Reserved)	-	0(0x00)
11(0X0B)	the Highest Limit Temperature	RD,WR	85(0x55)
12(0X0C)	the Lowest Limit Voltage	RD,WR	60(0X3C)
13(0X0D)	the Highest Limit Voltage	RD,WR	190(0xBE)
14(0X0E)	Max Torque(L)	RD,WR	255(0XFF)
15(0X0F)	Max Torque(H)	RD,WR	3(0x03)
16(0X10)	Status Return Level	RD,WR	2(0x02)
17(0X11)	Alarm LED	RD,WR	4(0x04)
18(0X12)	Alarm Shutdown	RD,WR	4(0x04)
19(0X13)	(Reserved)	RD,WR	0(0x00)
20(0X14)	Down Calibration(L)	RD	?
21(0X15)	Down Calibration(H)	RD	?
22(0X16)	Up Calibration(L)	RD	?
23(0X17)	Up Calibration(H)	RD	?

Tabla 2-1. Tabla de Control Zona EEPROM

❖ **ÁREA RAM:**

24(0X18)	Torque Enable	RD,WR	0(0x00)
25(0X19)	LED	RD,WR	0(0x00)
26(0X1A)	CW Compliance Margin	RD,WR	0(0x00)
27(0X1B)	CCW Compliance Margin	RD,WR	0(0x00)
28(0X1C)	CW Compliance Slope	RD,WR	32(0x20)
29(0X1D)	CCW Compliance Slope	RD,WR	32(0x20)
30(0X1E)	Goal Position(L)	RD,WR	[Addr36]value
31(0X1F)	Goal Position(H)	RD,WR	[Addr37]value
32(0X20)	Moving Speed(L)	RD,WR	0
33(0X21)	Moving Speed(H)	RD,WR	0
34(0X22)	Torque Limit(L)	RD,WR	[Addr14] value
35(0X23)	Torque Limit(H)	RD,WR	[Addr15] value
36(0X24)	Present Position(L)	RD	?
37(0X25)	Present Position(H)	RD	?
38(0X26)	Present Speed(L)	RD	?
39(0X27)	Present Speed(H)	RD	?
40(0X28)	Present Load(L)	RD	?
41(0X29)	Present Load(H)	RD	?
42(0X2A)	Present Voltage	RD	?
43(0X2B)	Present Temperature	RD	?
44(0X2C)	Registered Instruction	RD,WR	0(0x00)
45(0X2D)	(Reserved)	-	0(0x00)
46[0x2E)	Moving	RD	0(0x00)
47[0x2F)	Lock	RD,WR	0(0x00)
48[0x30)	Punch(L)	RD,WR	32(0x20)
49[0x31)	Punch(H)	RD,WR	0(0x00)

Tabla 2-2. Tabla de Control Zona RAM

Una peculiaridad de los actuadores es que tienen dos modos de funcionamiento, el modo servo motor y el llamado "modo rueda" o de giro continuo.

- **Modo Rueda o Giro Continuo:** Cuando se usa los motores AS-12A en este modo, los registros de posición carecen de utilidad, y solo se usa el de "Goal Speed". En este registro se podrá escribir un valor entre 0 y 2047 (0xFFF). Cada unidad es aproximadamente un 0.1% del par máximo que puede ejercer el motor. Si se usa un valor comprendido entre 0 y 1023, girará en sentido antihorario, siendo el valor 0 la parada. Si usamos el rango de valores de 1024 a 2047, girará en sentido horario. Esto se traduce a que el último bit se convierte en el que especifica la dirección de giro.

Ejemplo 2-1. Modo Rueda: si se escribe el valor 512 en el registro de "Goal Speed", el motor girará a un 50% del par máximo.

- **Modo Servo Motor:** para el uso de este modo, usaremos dos registros, "Goal Position" y "Goal Speed". En esta ocasión, "Goal Speed" solo tomará valores comprendidos entre 0 y 1023. Cada unidad tiene un valor aproximado de 0.111 rpm, por lo que, en este caso, en dicho registro si controla la velocidad (no como en el caso anterior, que puede llevar a confusión). Por otro lado, tenemos el registro de "Goal Position". Como se puede observar en la figura 2-7, existe una zona muerta de 60 grados, que es inaccesible para el motor (debido a su encoder). Los 300 grados restantes se codifican con un rango de valores de 0 a 1023.

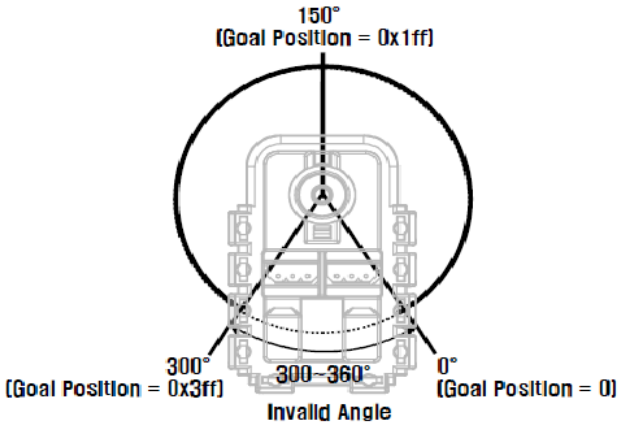


Figura 2-7. Explicación Registro "Goal Position"

Por tanto, si se usa este modo se deberá especificar la posición de destino deseada, así como a la velocidad a la que irá a dicha posición.

Ejemplo 2-2. Modo Servo Motor: Si queremos que el motor se posicione a 150º, se deberá escribir el valor 512 en el registro "Goal Position". Si queremos que se mueva a 33.3 rpm, se pondrá el valor 300 en el registro "Goal Speed".

La conclusión que se puede sacar acerca de estos actuadores es su versatilidad y la gran cantidad de información que se puede obtener de ellos. Como aspecto negativo, el modo de giro continuo tiene limitaciones, ya que no se puede controlar la velocidad sino el par, lo que genera una problemática si se quiere hacer un control en velocidad.[4]

[4] Se ha conseguido implementar un control en velocidad haciendo uso del modo Servo Motor. Se expondrá en el capítulo 5 apartado 3.

2.2.3 Piezas de Ensamble de Motores (Robotis).

Como se puede ver en el Anexo I, el kit dispone de una gran variedad de piezas que permiten diseñar múltiples eslabones para la conexión física de los motores. Sin embargo, son dos de estas piezas las principales para el diseño. La figura 2-8 es un extracto del datasheet de los AX-12A, donde se hace referencia a las dos piezas de ensamble principales, así como las diferentes formas de unirlas a los motores.

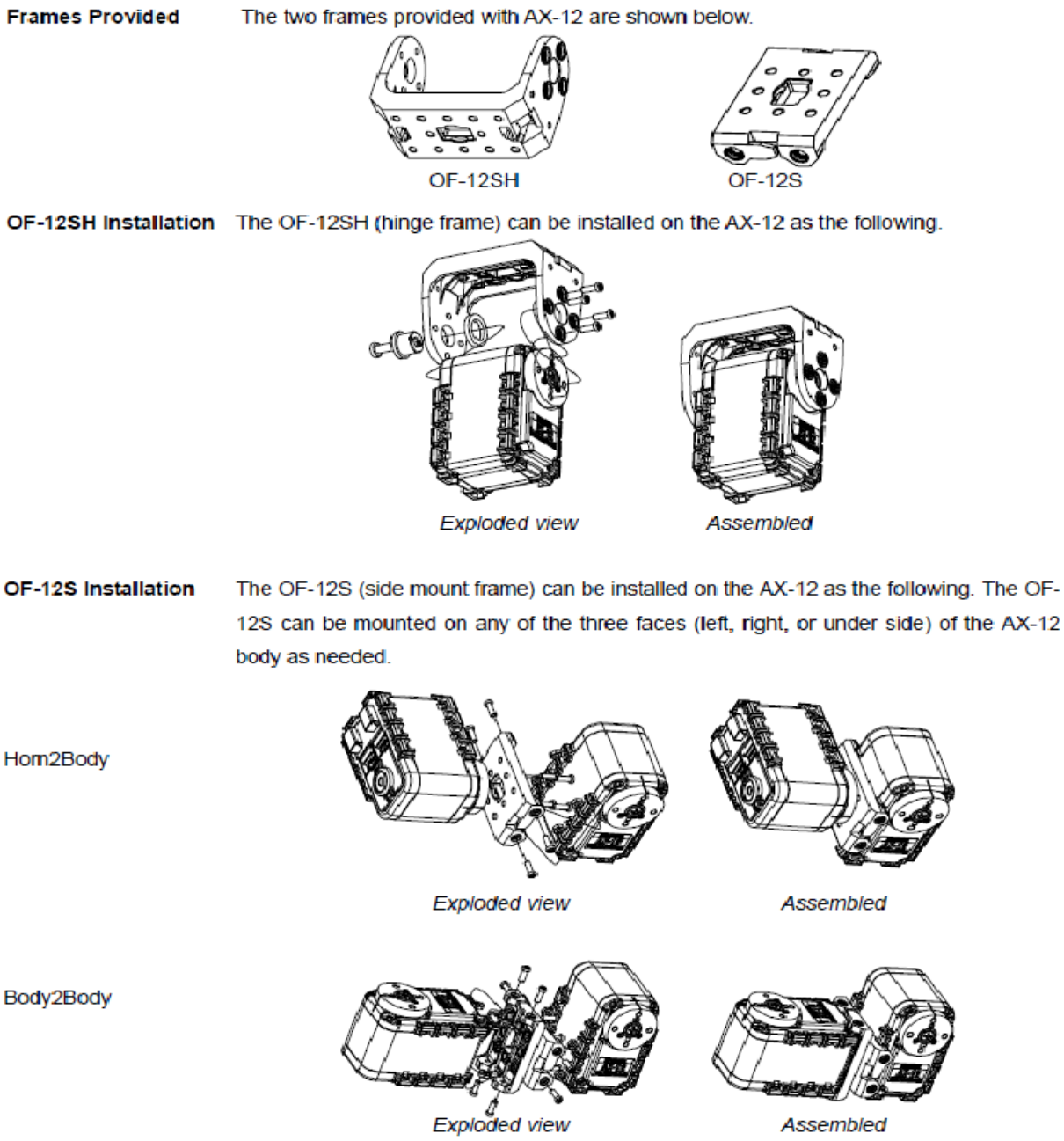


Figura 2-8. Piezas de Ensamble Motores

2.3 Procesadores.

2.3.1 Arbotix-M (Robocontroller).

Es un controlador compatible con Arduino, sompartiendo su lenguaje de programación, librerías y su IDE. Se beneficia de una comunidad de librerias de Código abierto y ejemplos. Desde su lanzamiento en 2010, se ha convertido en el favorito para el control de los actuadores de Dynamixel, siendo usado tanto para la enseñanza como para la investigación de alto nivel.

Figura 2-9. Arbotix - M

El Arbotix (Figura 3-1) cuenta con:[5]

- Microcontrolador AVR 16MHz (ATMEGA644p).
- Dos tomas de corriente continua para alimentación.
- 3 puertos Dynamixel (TTL).
- Regulador de 5 V. Puede trabajar con entradas de entre 7 y 30 V, y pudede suministrar 1.5A a 5V.
- Puerto Serie / Programación FTDI.
- Puerto de programación ISP.
- 20 puertos de E/S digitales (3 pines por entrada: Signal – VCC - Ground), de los cuales 4 están habilitados para PWM.
- 8 puertos de entradas analógicas.

Se ha decidido usar este controlador por tener un procesador bastante más potente que el de la controladora CM-530, además de una mayor accesibiliadd y versatilidad con los puertos que posee.

Sin embargo, la mayor ventaja que dispone es la posibilidad de programarlo a través de la IDE de Arduino, así como haciendo uso de su lenguaje de programación. Esto se debe a que la casa TrossenRobotis ha diseñado unas bibliotecas específicas para este microcontrolador que permite adentrarnos en las entrañas del Dynamixel, proporcionando la posibilidad de explotar todo su potencial.

[5] Para más información sobre el Arbotix-M, visitar el siguiente enlace (pagina oficial): http://learn.trossenrobotics.com/arbotix/arbotix-getting-started/38-arbotix-m-hardware-overview#&panel1-1

2.3.2 RaspBerry Pi 3 B:

Para la realización del sistema de visión artificial se ha usado una RaspBerry Pi 3B. Estos dispositivos son ordenadores de bajo costo y gran potencia. Son usadas tanto para investigación como para aprendizaje, teniendo un amplio abanico de posibilidades de programación.

Las características de las RaspBerry son las siguientes:

Figura 2-10. RaspBerry Pi 3B

- Procesador:
  - Chipset Broadcom BCM2387.
  - 1,2 GHz de cuatro núcleos ARM Cortex-A53
- GPU
  - Dual Core VideoCore IV ® Multimedia Co-procesador. Proporciona Open GL ES 2.0, OpenVG acelerado por hardware, y 1080p30 H.264 de alto perfil de decodificación.
  - Capaz de 1 Gpixel / s, 1.5Gtexel / s o 24 GFLOPs con el filtrado de texturas y la infraestructura DMA
- RAM: 1GB LPDDR2.
- Conectividad
  - Ethernet socket Ethernet 10/100 BaseT
  - 802.11 b / g / n LAN inalámbrica y Bluetooth 4.1 (Classic Bluetooth y LE)
  - Salida de vídeo
    - HDMI rev 1.3 y 1.4
    - RCA compuesto (PAL y NTSC)
  - Salida de audio
    - jack de 3,5 mm de salida de audio, HDMI
    - USB 4 x Conector USB 2.0
  - Conector GPIO
    - 40-clavijas de 2,54 mm (100 milésimas de pulgada) de expansión: 2x20 tira
    - Proporcionar 27 pines GPIO, así como 3,3 V, +5 V y GND líneas de suministro
  - Conector de la cámara de 15 pines cámara MIPI interfaz en serie (CSI-2)
  - Pantalla de visualización Conector de la interfaz de serie (DSI) Conector de 15 vías plana flex cable con dos carriles de datos y un carril de reloj
  - Ranura de tarjeta de memoria Empuje / tire Micro SDIO

2.4 Impresora 3D.

Uno de los objetivos principales del proyecto es el diseño y la fabricación de los dos brazos de robots. Para ello se ha usado el software de diseño Catia, así como una impresora 3D para la impresion de las piezas.

La impresora que se ha usado para la fabricación de las piezas es la Geeetech Prusa i3 Pro-W[6]. Es una impresora de bajo coste, la cual hay que montar y equilibrar antes de usarla. Sus características principales son:

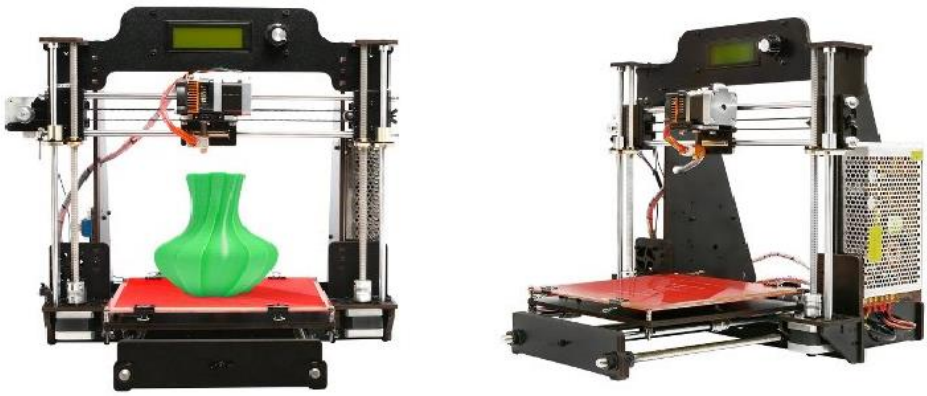


Figura 2-11. Impresora Geeetech Prusa i2 Pro - W

- Tecnología: FFF / FDM
- Tamaño de impresión. 200 x 200 x 180 cm
- Resolución de capa: 0,01 – 0,03 mm
- Precisión: 0,01 – 0,03 mm
- Diámetro de la boquilla: 0,03mm
- Tipo de filamento: ABS/PLA/Flexible PLA/NYLON/WOOD.
- Diámetro del filamento: 1,75 mm
- Temperatura cama caliente: 110º max.
- Temperatura Extrusor: 240º max.
- Dimensiones: 45 x 44 x 45 cm
- Compatibilidad: Win/Mac/Linux
- Software: descarga desde Geeetech, Easyprint 3D.
- Formato de los archivos: STL / G-code
- Panel LCD
- Fuente de alimentación: entrada 110-220V / salida DC12V/ 15A
- Conectividad: USB / Tarjeta SD / Wi-fi (módulo aparte)
- Estructura Física: Rep-rap
- Cuerpo de madera de 6 mm, color negro.
- Plataforma de construcción: Vidrio + Calefacción.
- Sistema de control: GT2560 de autodesarrollo, compatible con la función de nivelación automática.
- Motores: Angulo 1, 8º / con 1/6 de micro-stepping.

[6] Se podrá ver el manual de impresión en el Anexo III.

3 SISTEMAS.

"Corta tu propia leña y te calentará dos veces"

- Henry Ford -

Continuaremos con la explicación teórica de nuestro sistema, el cuál estará compuesto por los dos brazos robóticos. El capítulo se comenzará con una presentación de ambos robots. Tras esto, se dispondrá de una información detallada de las principales características de cada uno de ellos expuestas en dos subapartados diferentes.

Cada uno de estos subapartados dispondrá de un bloque de diseño y otro de estudio cinemático. Si se desea adquirir más información sobre su construcción o el método usado para la impresión, el lector podrá obtenerla en los anexos de la memoria.

3.1 Introducción.

Como se expuso al inicio del documento, los objetivos principales del proyecto giran alrededor del diseño y creación de dos brazos robóticos, los cuales compondrán el sistema. Para ello se creó primero un modelo de robot industrial de 6 grados de libertad, sobre el cuál se han realalizado los experimentos de control. Por tanto, este brazo debía de ser suficientemente robusto para la experimentación.

El robot Compliant debe de tener articulaciones pasivas flexibles, pero que fueran capaces de sobortar la experimentación.

El resultado del diseño y contrucción de ambos brazos se muestran en las siguientes imágenes:

Figura 3-2. Robot Industrial 6 GDL

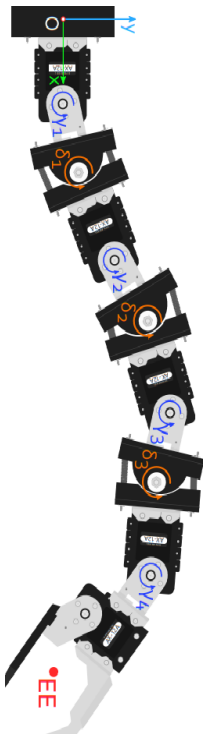


Figura 3-1. Robot Compliant

3.2 Brazo Robótico Industrial 6 Grados De Libertad.

Se comenzará con el barzo robótico industrial de seis grados de libertad. En un principio, fue el objetivo principal del proyecto, y en el cuál se invirtió buena parte del tiempo para su diseño, realizándose diferentes modelos, todos ellos haciendo uso únicamente de las piezas del kit.

Tras haber probado diferentes prototipos, se comprueba que, si se desea hacer un brazo robótico industrial de seis grados de libertad, las dimensiones que alcanza el robot son demasiado grandes y, por tanto, las articulaciones inferiores sufren debido a la gran cantidad de esfuerzo que deben soportar, impidiendo que realicen su cometido. Este problema se soluciona poniendo dos motores en paralelo en las articulaciones que necesitan un mayor par motor para mover el conjunto, lo que implica rediseñar el robot al completo, y buscar nuevas piezas que permitan ensamblar ambos actuadores en paralelo.

Tomada la decisión de buscar piezas extras al kit, se procede al diseño de las piezas y su posterior impresión 3D. Finalizada la impresión de todas las partes del robot, se procede al montaje y estudio matemático, y tras esto, a la decisión del tipo de control que se realizará, sus simulaciones y su posterior programación en el controlador Arbotix-M.

3.2.1 Diseño del Robot.

Finalmente, se decide realizar un robot industrial antropomórfico (también llamado angular), que implica que todas las articulaciones son rotativas, condición de diseño evidente debido a la naturalez de los servomotores que se usarán como actuadores. Gracias a los seis grados de libertad que poseerá el robot, se podrá situar la herramienta en el punto del espacio deseado (px, py, pz) y en la orientación adecuado (roll, pich y yaw) adecuados para la manipulacción correcta del objetivo.

Para determinar la orientación de las articulaciones se requiere que pueda realizar un desacople cinemático entre el brazo y la muñeca, lo que simplificará el estudio del robot. Para ello, la muñeca deberá ser esférica.

A continuación, se expondrán las imágenes realizadas en Catia para la impresión 3D de cada una de las piezas del robot. En las figuras se puede ver marcado en naranja el contorno de la pieza. Todas las piezas están diseñadas sobre una plataforma ovalada o circular, que tiene como cometido mejorar la impresión 3D y evitar probemas de Wraping.

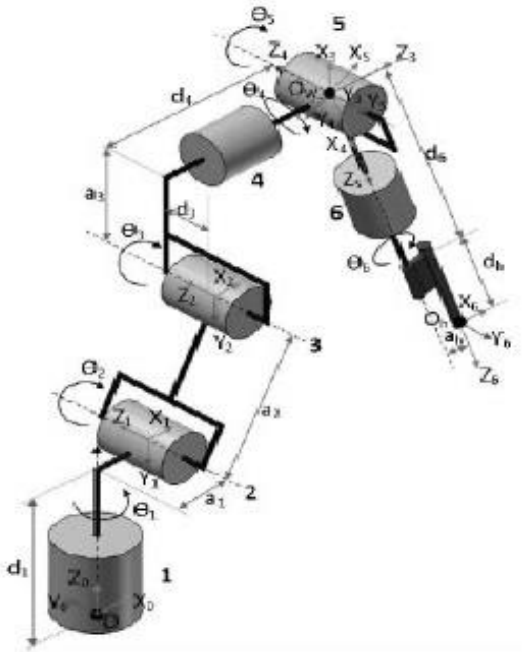


Figura 3-3. Esquema Robot Antropomórfico

I. Esalbón 0: Base.

Es la estructura sobre la cuál se apoyará el eslabón 1. Está compuesta por dos piezas octogonales. La inferior tiene un hueco circular, en el que estará el primer motor. Se une a la segunda parte por 8 tornillos con dos tuercas, que servirán para nivelar ambos módulos y queden a la altura necesaria. La parte superior de la estructura tiene un canal circular, en el cuál se añadiran rodamientos para que gire el eslabón 1.

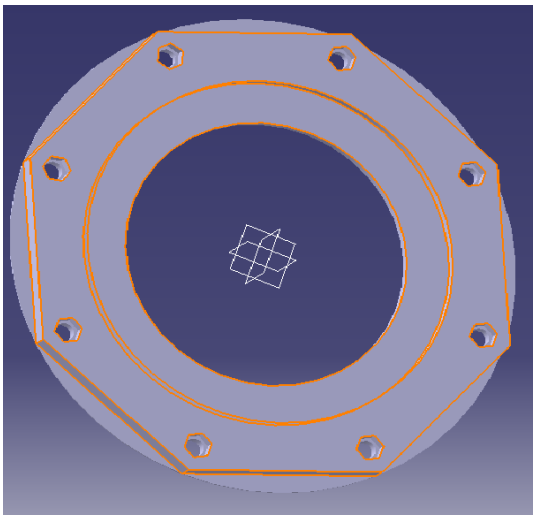


Figura 3-4. Elabón 0. Pieza Inferior

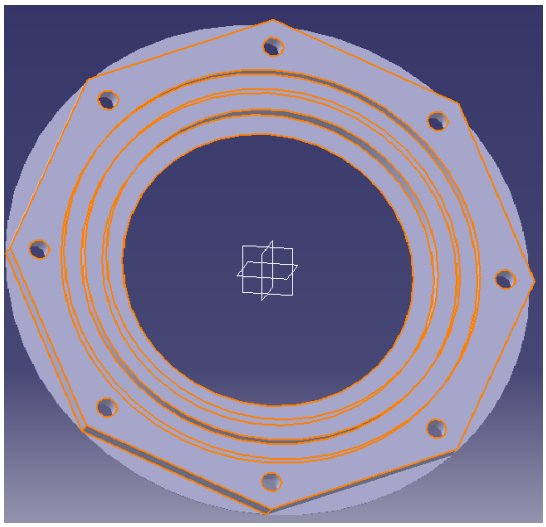


Figura 3-5. Eslabón 0. Pieza superior

II. Esalbón 1: Plataforma Circular. Unión de las articulaciónes 1ª y 2ª.

El eslabón 1 está compuesto de una pieza circular que se posará encima de la plataforma de rodamientos sobre la cuál girará. Es la unión de la primera y la segunda articulación. Cuenta con 8 orificios para atornillar los tres motores que conecta, uno para la primera articulación, y dos de la segunda. Cuenta con un hueco por el que pasará el bus de comunicación de los actuadores.

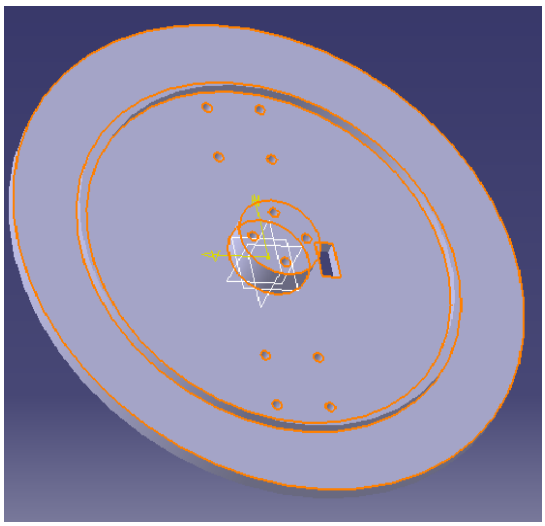


Figura 3-6. Eslabón 1. Cara Superior

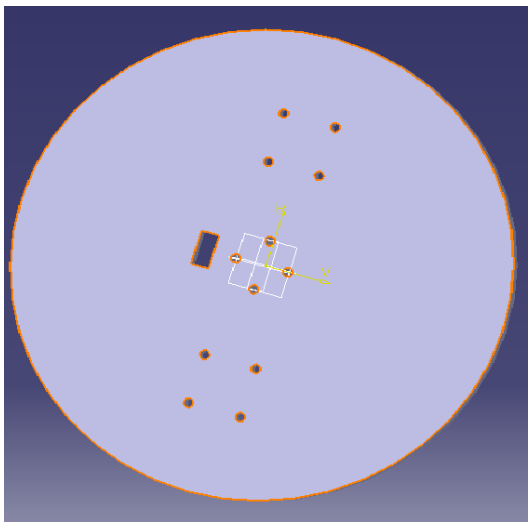


Figura 3-7. Eslabón 1. Cara Inferior

III. Esalbón 2: Unión de las articulaciones 2ª y 3ª.

Este eslabón es el más grande de todos y, por tanto, el que mas esfuerzos soporta. Por este motivo, se ha diseñado una estructura rectangular muy consistente, con piezas anchas y entramado triangular para dar consistencia y rigidez. Consta de dos piezas diferentes para formar el rectángulo y una central que sirve de apoyo a la estructura. Las describiremos una a una a continuación:

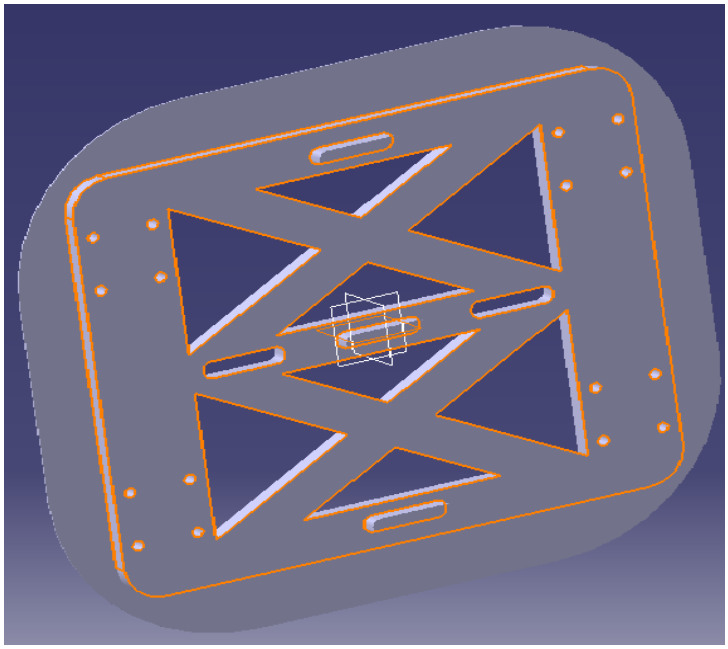


Figura 3-8. Eslabón 2. Pieza Superior e Inferior

Piezas Superio e Inferior:

Esta pieza está diseñada para atornillar cuatro motores, dos correspondientes a la segunda articulación y otros dos de la tercera. En los extremos y en el centro tienen unos pasantes que sobresalen, e irán introducidos en las piezas laterales y en la central. Como la pieza central no estará atornillada a nada, se han hecho dos alojamientos para que encajen sus pasadores y quede bien fijadas gracias a la presión de las demás piezas.

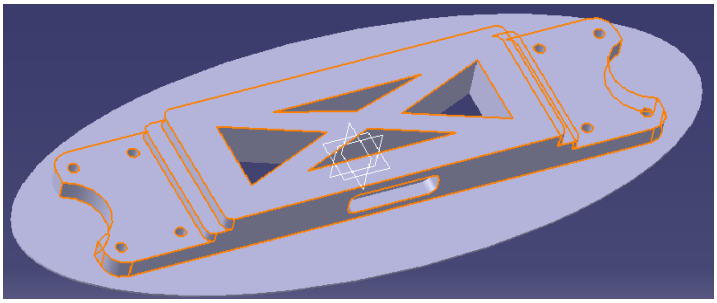


Figura 3-9. Eslabón 2. Piezas Laterales

Piezas Laterales:

Estos segmentos se atornillan a un motor de cada una de las articulaciones que conecta. Tiene dos alojamientos para introducir el pasador de las caras superior e inferior. Disponen de unos rebajes en los extremos para dejar hueco a la pieza móvil del motor.

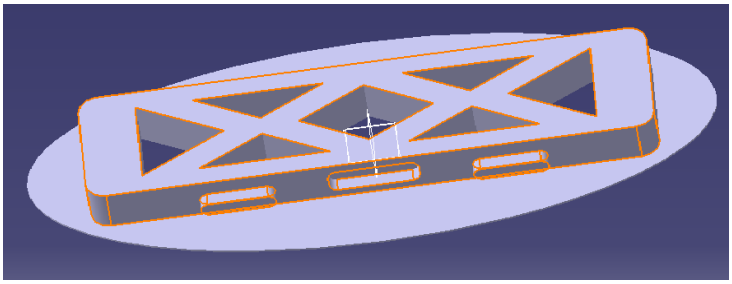


Figura 3-10. Eslabón 2. Pieza Central

Pieza Central Interior:

Este segmento va encajado a presión por los cuatro tetones y los dos alojamientos que dispone. Es una pieza de refuerzo que ayuda a repartir el esfuerzo de las piezas laterales, dando rigidez a la estructura.

IV. Eslabón 3: Unión del Brazo y la Muñeca.

Este segmento está diseñado para unir la tercera y cuarta articulación. Las partes móviles de los dos motores de la tercera articulación van atornillados con cuatro tornillos penetrantes cada uno. El cuerpo del Dynamixel de la cuarta articulación va atornillado usando ocho tornillos, dejando la parte trasera del motor (donde se conectan los buses) en el hueco de la pieza.

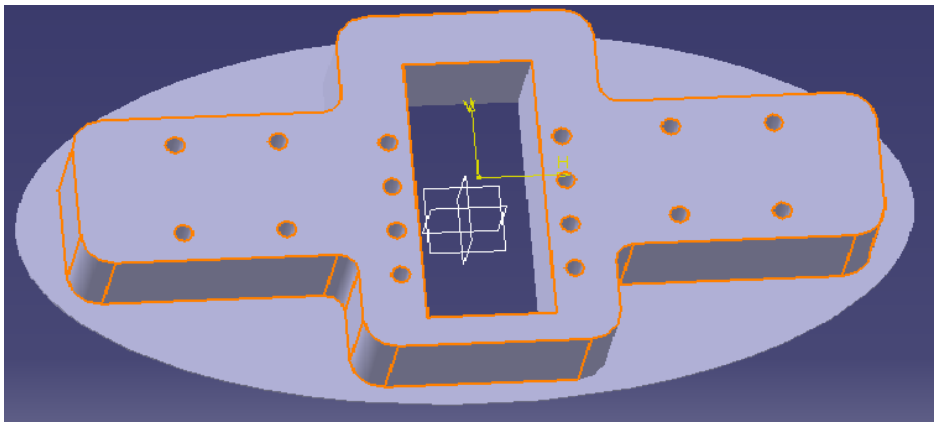


Figura 3-11. Eslabón 3. Unión Brazo – Muñeca

V. Pinza:

La última pieza por diseñar es el efector, la cuál se decidió realizar de manera simple y funcional, ya que no entraba dentro de los objetivos el realizar una herramienta compleja, si no una que permitiera recrear los experimentos. Por tanto, se decide que la pinza será una placa plana que irá atornillada a la parte móvil del motor.

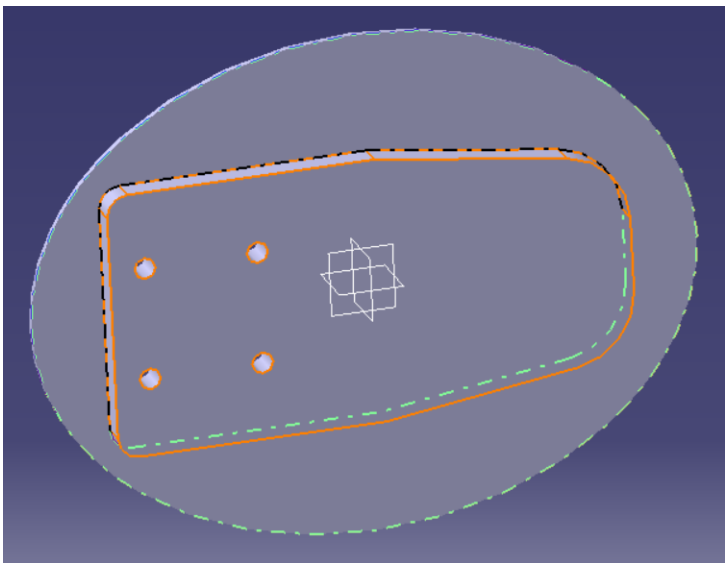


Figura 3-12. Pinza.

3.2.2 Construcción Brazo Industrial 6 GDL.

A continuación, se expondrán unas series de pasos a seguir con imágenes y una breve explicación del procedimiento que hay que seguir para montar el brazo robótico industrial.

1. Coger un tabón de aglomerado de 20x20 cm. Colocar eslabón 0 y taladrar para poder atornillar la base.
2. Una vez hechos todos los taladros, pasar los tornillos, con la cabeza por la parte de debajo de la madera. Colocar una primera tuerca en la mordura del eslabón 0 parte inferior. Tras esto, colocar una tuerca y una arandela en cada tornillo a la altura del eslabón 1 aproximadamente. Colocar encima de las arandelas el eslabón 0 parte superior, y sujetar con otra arandela y otra tuerca por tornillo. Una vez fijados todos los tornillos, se pueden colocar los rodamientos.
3. Se atornilla por la parte inferior del eslabón 1 el motor y los fijamos.
4. Una vez unido el primer motor con el eslabón 1, se coloca encima del eslabón 0 y se va bajando las tuercas y contratuercas hasta que la base del motor uno toque con la madera. Una vez en esta posición, se aprieta y con un lápiz se marca para poderla fijarla.
5. Quitar el eslabón 1. Se desacopla la base del motor y se posiciona encima de la silueta que se ha pintado en el paso anterior. Marcar los agujeros donde irán los tornillos.
6. Se fijan las piezas que servirán de articulación para el segundo grado de libertad.
7. tornillar la base del motor 1 a la madera y se nivela el eslabón 1 haciendo uso de las tuercas y contratuercas. Buscar que tenga una presión adecuada sobre los rodamientos y permita un giro suave. Se conecta el bus al primer motor y se saca por el orificio destinado a ello.
8. Fijar los otores de la segunda articulación al eslabón 1 y conectamos los busses.
9. Colocar en las caras superior e inferior del eslabón 2 las piezas en las cuales se atornillarán los motores de la segunda y tercera articulación.
10. Las piezas se colocan en los motores inferiores y se fijan. Tras esto, se cogen ambas caras laterales y se colocan en la posición adecuada encajándolas con los motores. Atornillar a los dos actuadores correspondientes de la segunda articulación.
11. Colocar los dos motores siguientes y fijar. Una vez ajustados, se coloca la pieza central en su posición encajando los teltones en sus orificios. Conectar los motores a través de los buses.
12. Colocar la ultima cara del eslabón 2. Atornillar a los motores para que queden bien apretados y compacto.
13. Colocar la parte móvil de la articulación 3 en os motores.
14. Unir el eslabón 3 al motor correspondiente a la cuarta articulación. Colocar dejando la parte del bus en la parte superior de la pieza
15. Fijar la parte fija que unirá la cuarta articulación con el resto de la muñeca.
16. Conectar el bus entre el tercer y el cuarto grado de libertad.
17. Atornillar el eslabón 2 y 3.
18. Atornillar el resto de la muñeca con la pinza ya colocada en su posción.

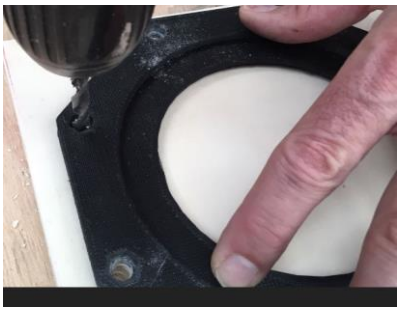


Figura 3-13. Montaje Robot Industrial Paso 1

Figura 3-14. Montaje Robot Industrial Paso 2

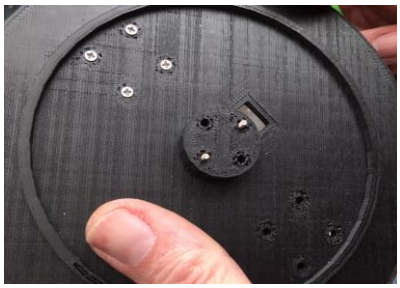


Figura 3-15. Montaje Robot Industrial Paso 3

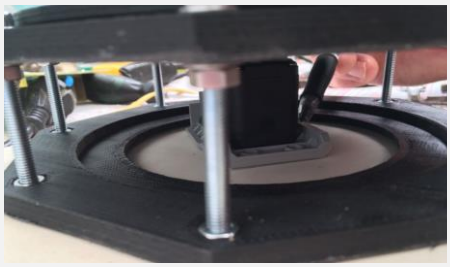


Figura 3-16. Montaje Robot Industrial Paso 4

Figura 3-17. Montaje Robot Industrial Paso 5

Figura 3-18. Montaje Robot Industrial Paso 6

Figura 3-19. Montaje Robot Industrial Paso 7

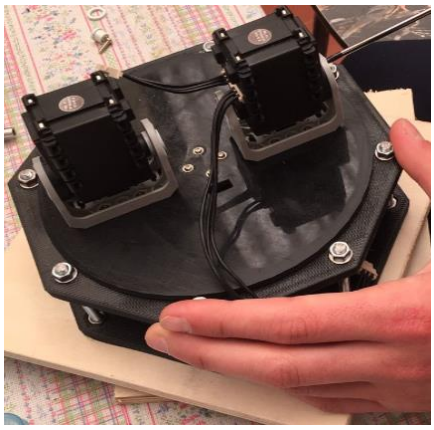


Figura 3-20. Montaje Robot Industrial Paso 8

Figura 3-21. Montaje Robot Industrial Paso 9

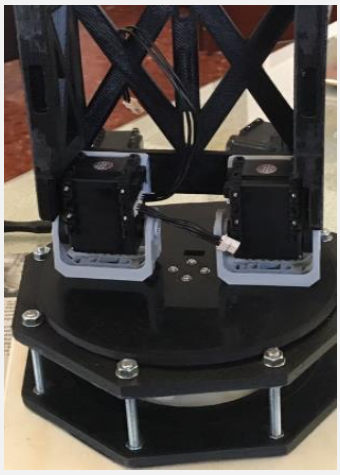


Figura 3-22. Montaje Robot Industrial Paso 10

Figura 3-23. Montaje Robot Industrial Paso 11

Figura 3-24. Montaje Robot Industrial Paso 12

Figura 3-25. Montaje Robot Industrial Paso 13

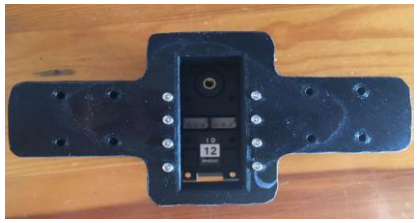


Figura 3-26. Montaje Robot Industrial Paso 14

Figura 3-27. Montaje Robot Industrial Paso 15

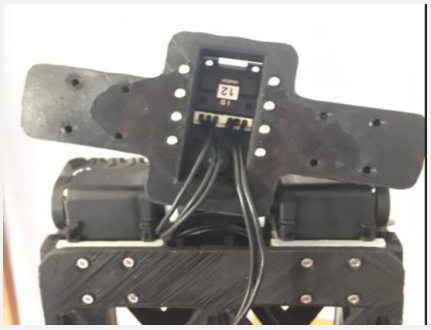


Figura 3-28. Montaje Robot Industrial Paso 16

Figura 3-29. Montaje Robot Industrial Paso 17

Figura 3-30. Montaje Robot Industrial Paso 18

3.2.3 Cinemática del Robot.

La cinemática de un robot estudia las posibilidades de movimiento que tiene. Un análisis completo engloba el estudio de las posiciones, velocidades y aceleraciones de cada uno de los elementos del robot, todos ellos sin tener en cuenta las fuerzas que actúan sobre el mismo.

Para realizar el estudio de la cinemática, se debe referir a cada una de las propiedades geométricas basadas en el tiempo que transcurre durante el movimiento. Se pueden distinguir dos tipos de cinemáticas:

- Cinemática Directa: Calcula la posición del extremo del robot con respecto a un sistema de coordenadas como referencia. Se usan una serie de ecuaciones que tienen en cuenta la medida de cada uno de los eslabones que lo componen, así como los valores específicos de cada una de sus articulaciones.
- Cinemática Inversa: Proceso inverso al anterior. Calcula los valores que deben de tener cada una de las articulaciones del robot para que el extremo de este se encuentre en un punto concreto y con una determinada orientación (si sus grados de libertad se lo permiten).

I. Cinemática Directa.

Para realizar la cinemática directa del brazo de robot industrial, lo primero que debemos hacer es construir el sistema de referencia de Denavit y Hartenberg. Es un método matricial que permite relacionar el sistema de referencia de las articulaciones mediante cuatro transformaciones básicas que depende únicamente de las características geométricas del robot. Las transformaciones son las siguientes:

1. Rotación alrededor del eje z_{i-1} un ángulo θ_i.
2. Traslación a lo largo de z_{i-1} una distancia d_i.
3. Traslación a lo largo de x_i una distancia a_i.
4. Rotación alrededor del eje x_i un ángulo α_i.

Para nuestro robot, los ejes de coordenadas se establecen como se observa en la figura.

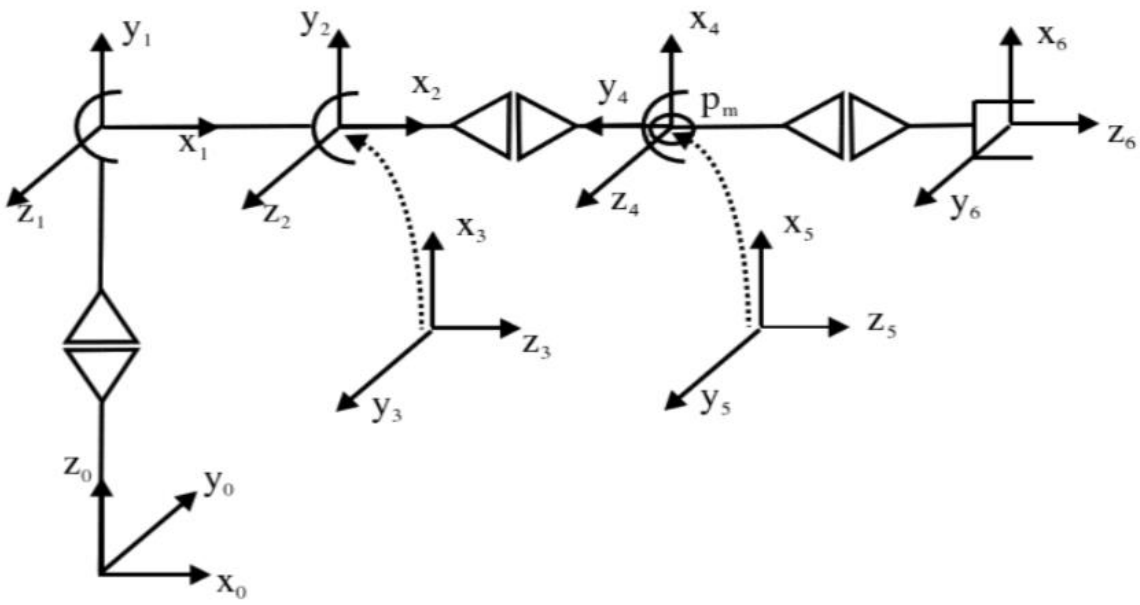


Figura 3-31. Ejes de Coordenadas Robot Industrial

Una vez definidos los ejes, construimos la tabla de parámetros de Denavit y Hartenberg. Se ha añadido una columna denominada offset[7], que se corresponde con un giro en la articulación para que coincida con la posición inicial del robot real.

Articulación	*θ (rad)*	*d (m)*	*a (m)*	*A (rad)*	*Offset (rad)*
Q1	θ_1	L1	0	π/2	π
Q2	θ_2	0	L2	0	π/2
Q3	θ_3	0	0	π/2	π/2
Q4	θ_4	L3	0	- π/2	0
Q5	θ_5	0	0	π/2	0
Q6	θ_6	L4	0	0	0

Tabla 3-1. Parámetros Denavit y Hatemberg Robot Industrial

Para el caso que nos ocupa, las longitudes tienen las siguientes medidas:

L1 (m)	**L2 (m)**	**L3 (m)**	**L4 (m)**
0.096	0.170	0.132	0.150

Tabla 3-2. Longitudes del Robot

Una vez están definidos todos los parámetros del brazo robótico, se deben crear las matrices de transformación homogéneas, que nos permitirán relacionar cada articulación con la siguiente, y tienen la siguiente estructura:

$$A_i^{i-1} = \begin{pmatrix} \cos\theta_i & -\cos\alpha_i \cdot \sin\theta_i & \sin\alpha_i \cdot \sin\theta_i & a_i \cdot \cos\theta_i \\ \sin\theta_i & \cos\alpha_i \cdot \cos\theta_i & -\sin\alpha_i \cdot \cos\theta_i & a_i \cdot \sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ecuación 3-1. Matriz de Transformación Homogénea.

Para cada una de las filas de la Tabla 3-1, se debe crear una matriz de transformación. Si se multiplican todas las matrices en el orden correcto, como muestra la Ecuación 3-2, se obtendrá la matriz T.

[7] Este dato de offset es encesario para ejecutar el archivo de Matlab CinematicaTool, el cuál hace uso del ToolBox de Peter Corke, que ha sido implementado para la simulación del brazo robótico.

$$T = A_1^0 \cdot A_2^1 \cdot A_3^2 \cdot A_4^3 \cdot A_5^4 \cdot A_6^5 = A_6^0 = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad con \quad a = n \times o$$

Ecuación 3-2. Matriz de Tranformación T.

Como resultado de la matriz T, se tienen los puntos $p_x, p_y \, y \, p_z$ que corresponden con las coordenadas cartesianas del extremo del robot, así como los vectores n, o y a que nos forman un sistema de ejes cartesianos, indicando la orientación.

II. Cinemática Inversa.

Como se explica al comienzo de este capitulo, se diseñó un robot antropomórfico al cuál se le puede aplicar el procedimiento de desacople cinemático, que será de utilidad en este apartado. Esto significa que se puede tratar de manera independiente los problemas de posicionamiento y de orientación del extremo.

Si se considera únicamente los tres primeros grados de libertad, observamos que tienen una estructura planar, es decir, que quedan contenidos en un mismo plano. Gracias a esto, usando métodos geométricos podemos obtener la posición del extremo del robot. Haciendo uso de los tres últimos grados de libertad, es posible averiguar la orientación del extremo sin necesidad de saber en que posición se encuentran el resto de las articulaciones.

i. Resolución del problema de Posicionamiento del Extremo del Robot por Métodos Geométricos:

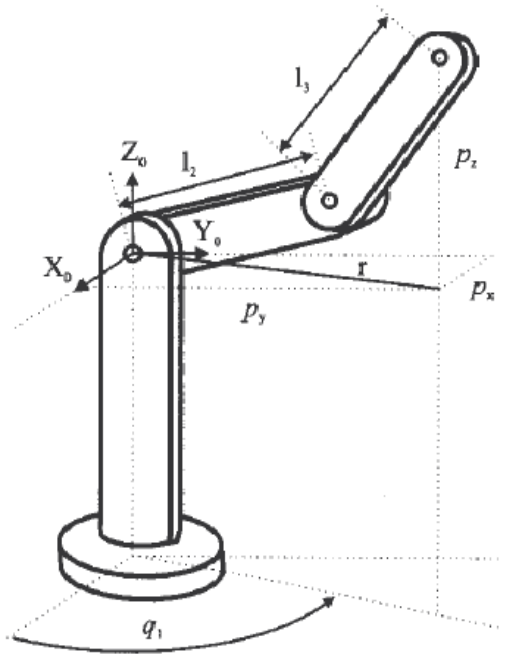


Figura 3-32. Esquema Brazo Robot

Identificado el primer problema, se tienen como datos los puntos cartesianos del extremo del robot, denominados $p_x, p_y \, y \, p_z$. Debido a la naturaleza planar del brazo, queda definido el valor de la primera articulación.

$$q_1 = \tan^{-1}\left(\frac{p_x}{p_y}\right)$$

Ecuación 3-3. Cinemática Inversa. Articulación 1.

Considerando los eslabones 2 y 3, y usando el teorema del coseno, obtendremos las siguientes ecuaciones.

$$r^2 = p_x^2 + p_y^2$$

$$r^2 + p_z^2 = L_2^2 + L_3^2 + 2L_2 \cdot L_3 \cdot \cos q_3$$

$$\cos q_3 = \frac{p_x^2 + p_y^2 + p_z^2 - L_2^2 - L_3^2}{2L_2 \cdot L_3}$$

Por motivos de eficiencia a la hora de la computación y la programación del algoritmo, es mejor usar la expresión de la arcotangente en lugar del arcoseno, obteniendo la ecuación 3-4.

$$q_3 = \tan^{-1}\left(\frac{\pm\sqrt[2]{1-cos^2 q_3}}{\cos q_3}\right) \qquad con\cos q_3 = \frac{p_x^2 + p_y^2 + p_z^2 - L_2^2 - L_3^2}{2L_2 \cdot L_3}$$

Ecuación 3-4. Cinemática Inversa. Articulación 3.

Llegados al cálculo de la articulación 2, se consideran dos posibles soluciones diferentes, comúnmente conocidas como Codo Abajo y Codo Arriba.

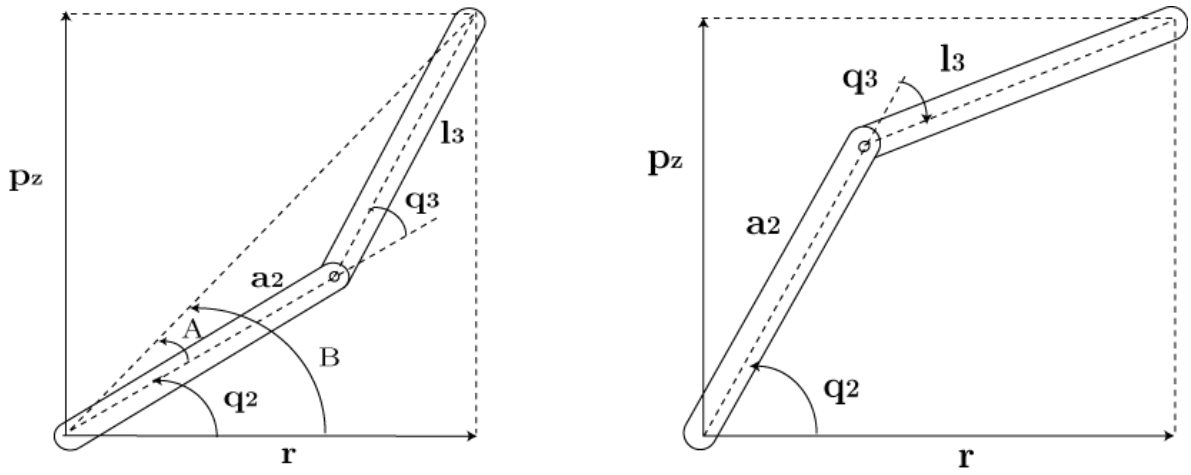


Figura 3-33. Codo Abajo y Codo Arriba.

El cálculo de la articulación 2 se hace mediante las variables A y B:

$$q_2 = B - A \quad con \quad B = \tan^{-1}\left(\frac{p_z}{r}\right) = \tan^{-1}\left(\frac{p_z}{\pm\sqrt{{p_x}^2 + {p_y}^2}}\right) \quad y \quad A = arcotan\left(\frac{L_3 \cdot \sin q_3}{L + L_3 \cdot \cos q_3}\right)$$

Sustituyendo, obtenemos la ecuación 3-5 que nos definirá el valor de la articulación 2.

$$q_2 = \tan^{-1}\left(\frac{p_z}{\pm\sqrt{{p_x}^2 + {p_y}^2}}\right) - arcotan\left(\frac{L_3 \cdot \sin q_3}{L + L_3 \cdot \cos q_3}\right)$$

Ecuación 3-5. Cinemática Inversa. Articulación 2.

Una vez obtenidas las ecuaciones 4-3, 4-4 y 4-5, se ha conseguido resolver el problema cinemático inverso para las tres primeras articulaciones que fijan la posición del extremo del robot en el espacio de trabajo.

ii. Resolución del problema de Orientación del Extremo del Robot:

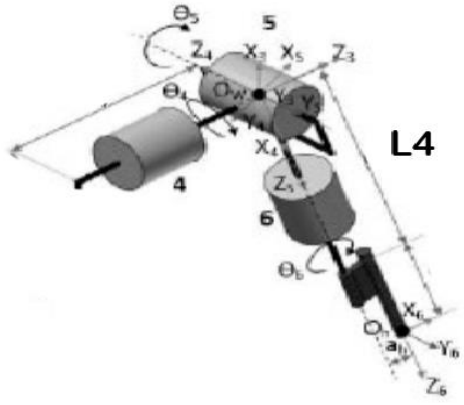


Figura 3-34. Muñeca del Robot.

Se comienza distinguiendo dos puntos diferentes, el conocido como "centro de la muñeca" (P_m), que corresponde con el origen del Sistema de referencia de la articulación 5, y el final del robot, que se corresponde con el origen del Sistema de la articulación 6 (P_r), y corresponde con los anteriormente conocidos puntos p_x, p_y y p_z.

$$p_r = [p_x, p_y, p_z]'$$

Como se observa en la figura, los ejes coordenados en Z de las articulaciones 5 y 6 deben coincidir, y están separadas una distancia igual a L_4, por lo que obtenemos que:

$$p_m = p_r - L_4 \cdot z_6 \quad con \quad z_6 = \left[a_x, a_y, a_z\right]^T$$

Una vez que se obtiene el punto p_m, se debe averiguar los valores de las articulaciones 4, 5 y 6 que determinarán la orientación deseada del efector del robot. Para ello se volverá a usar las matrices de transformación. Partiendo de la matriz R_6^0 (submatriz de rotación de T_6^0) se obtiene:

$$R_6^0 = [\, n \quad o \quad a] = R_3^0 \cdot R_6^3 \qquad con \qquad R_3^0 = A_1^0 \cdot A_2^1 \cdot A_3^2$$

Ecuación 3-6. Matrices de Rotación

Si despejamos la matriz R_6^3 de la ecuación 3-6, se obtiene una matriz que solo dependerá de los valores específicos de los tres últimos grados de libertad. En la siguiente ecuación se expondrá cuál es el resultado simbólico de realizar dicha ecuación.

$$R_{6_{i,j}}^3 = \begin{bmatrix} \cos q_4 \cos q_5 \cos q_6 - \sin q_4 \sin q_6 & -\cos q_4 \cos q_5 \sin q_6 - \sin q_4 \cos q_6 & \cos q_4 \sin q_5 \\ \sin q_4 \cos q_5 \cos q_6 + \cos q_4 \sin q_6 & -\sin q_4 \cos q_5 \sin q_6 + \cos q_4 \cos q_6 & -\sin q_4 \cos q_5 \\ -\sin q_5 \cos q_6 & \sin q_5 \sin q_6 & \cos q_5 \end{bmatrix}$$

Ecuación 3-7. Matriz de Rotación Muñeca.

De las relaciones expresadas en esta matriz, podemos despejar los valores de q_4, q_5 y q_6, obteniendo las siguientes expresiones:[8]

$$q_4 = \arcsin\left(\frac{R_{6_{2,3}}^3}{R_{6_{3,3}}^3}\right) \qquad q_5 = arcos\left(R_{6_{3,3}}^3\right) \qquad q_6 = arcotan\left(-\frac{R_{6_{3,2}}^3}{R_{6_{3,1}}^3}\right)$$

Ecuación 3-8. Coordenadas Articulares Muñeca

[8] Como en el caso de la cinemática directa para la articulación 3, se realizarán los cambios pertinentes para usar la función arcotangente, debido a su naturaleza más robusta a la hora de la computación.

3.2.4 Generador de Trayectorias – Jacobiano.

Existe la posibilidad de establecer relación entre las deribadas de las posiciones de cada uno de los grados de libertad del robot. Esto permitirá conocer qué velocidad debe establecerse en cada instante de tiempo en cada articulación, para que el extremo del robot consiga llegar a la posición adecuada. Esta relación se obtiene a través de la denominada matriz Jacobiana.

La matriz Jacobiana directa permite conocer la velocidad del extremo del robot haciendo uso de las velocidades de cada grado de libertad. Como en el caso de la cinemática, este método tiene su inverso, que proporcionará unas velocidades determinadas para el extremo del robot.

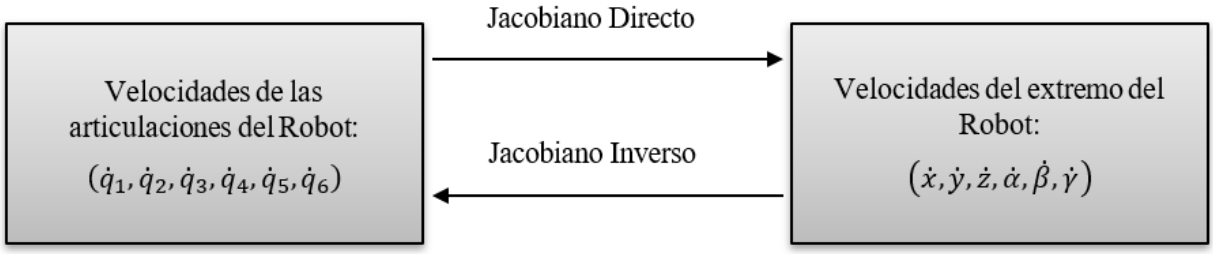


Figura 3-35. Esquema Jacobiano.

Si se ha resuelto el problema cinemático, es fácil obtener el jacobiano, realizando las derivadas como se puede observar la la ecuación 3-9.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} = J \cdot \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \\ \dot{q_3} \\ \dot{q_4} \\ \dot{q_5} \\ \dot{q_6} \end{bmatrix} \qquad con \qquad J = \begin{bmatrix} \frac{\delta f_x}{\delta q_1} & \cdots & \frac{\delta f_x}{\delta q_6} \\ \vdots & \ddots & \vdots \\ \frac{\delta f_\gamma}{\delta q_1} & \cdots & \frac{\delta f_\gamma}{\delta q_6} \end{bmatrix}$$

Ecuación 3-9. Matriz Jacobiana.

Debido a que el valor numérico de cada elemento de la matriz será diferente en cada instante según la posición en la que se encuentre, el valor de la jacobiana será diferente para cada uno de los puntos del espacio de trabajo.

Existen problemas de singularidades de un robot, y estas surgen cuando el jacobiano se hace nulo, lo que implica que un pequeño incremento de las coordenadas cartesianas provocaría un incremento infinito en las coordenadas articulares. Esto provocaría que los actuadores del robot intentasen dar unas velocidades imposibles de alcanzar por el extremo del robot, así como perjudicial para el sistema robótico al completo.

Para evitar estas singularidades, se realizará un proceso matemático que incluirá un termino externo variable, que impedirá que la matriz jacobiana se anule y provoque estos errores, pero se explicará en el siguiente apartado del capitulo.

3.3 Brazo de Robot Compliant.

Este brazo robótico es muy diferente a los que se acostumbran a ver debido a que no se ha buscado una configuración común en la industria, sin embargo, su propósito es poder probar un nuevo tipo de control que permite incluir articulaciones flexibles.

Este tipo de robot está pensado para ir en drones y poder realizar tareas aéreas. Debido a la naturaleza en la que estará funcionando, las amortiguaciones flexibles permitirán ejercer presión de manera controlada, así como en caso de que se produzca un choque entre el brazo de robot y algún animal u objeto que esté en el espacio de trabajo del robot, la amortiguación pueda absorber parte del impacto y, en medida de lo posible, no desestabilice al dron.

Estas articulaciones flexibles se consideran grados de libertad, pero no se puede ejercer una actuación sobre ellos, pero si se debe conseguir información a cerca de su posicionamiento. Por este motivo, se ha necesitado realizar un sistema de visión artificial que permita reconocer el ángulo que existe entre las amortiguaciones.

Para simplificar el problema, se ha realizado un robot planar, teniendo así una altura de la pinza constante, y sus desplazamientos se limitan a dos dimensiones.

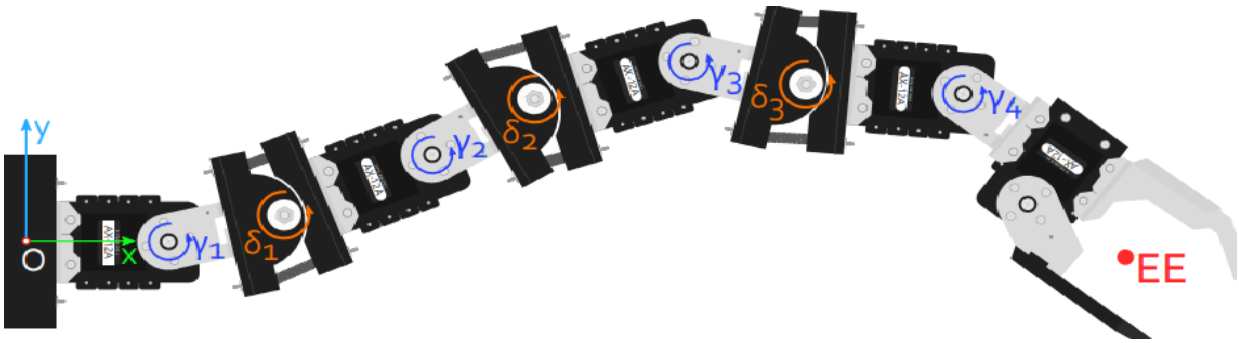


Figura 3-36. Robot Compliant

3.3.1 Diseño de Piezas:

El diseño de este brazo de robot será simple, ya que todas las piezas serán iguales y se busca únicamente la conexión entre los motores, todos ellos en la misma posición. La dificultad que se encuentra es la poca superficie que tienen los actuadores para conectarse entre ellos, lo que implicó crear una base cuya función únicamente es aumentar la zona de unión y disponer de espacio suficiente para poder hacer un sistema que permita la amortiguación. Una vez se dispuso de espacio suficiente, se pudo realziar la articulación. Como en las piezas anteriores, los diseños están realizados sobre una base en forma de elipse, que servirá para mejorar la calidad de la impresión y evitar problemas de Wrapping.

I. Base de motores:

La base es rectangular, y tiene unas dimensiones de 60 x 40 mm. Dispone de cuatro orificios en el centro, que se usarán para unirlo a las piezas de ensamble con los motores, así como de otros cuatro agujeros colocados en las esquinas, que permitirán unir la base al resto de la articulación.

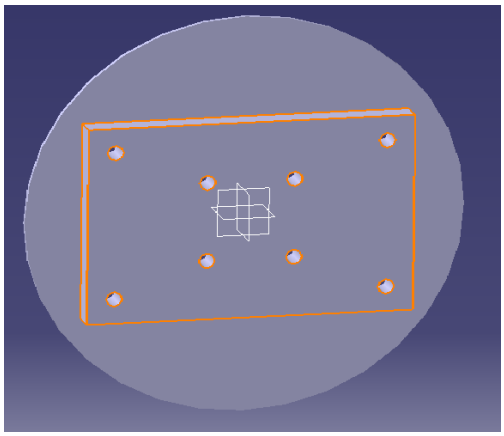


Figura 3-37. Base de Motores

II. Unión flexible:

Esta union cuenta con dos partes, cada una correspondiente a uno de los motores. La del primer motor (el que tiene la union en la parte móvil), cuenta con dos semicirculos colocados en la parte superior e inferior, mientras que la pieza de unión del segundo actuador cuenta con una sola circunferencia central. Esto permitirá unir ambas piezas como si de un "sandwich" se tratase, y tras esto colocar un pasador que los una. Para la amortiguación, ambas estructuras cuentan con cuatro agujeros en los cuales irán cuatro muelles que servirán como amortiguadores, así como otros cuatro en agujeros en las esquinas que permiten unirlo con la base.

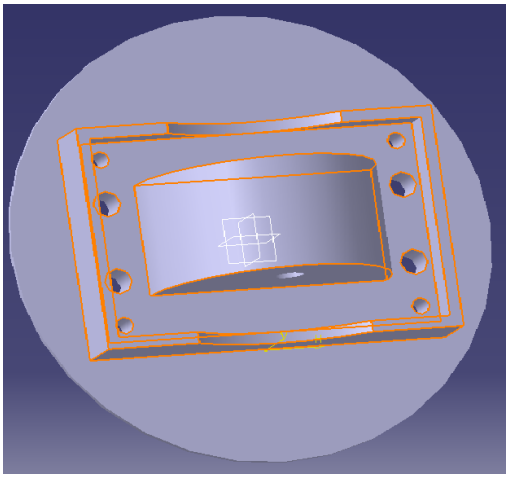


Figura 3-38. Pieza Unión Flexible 1

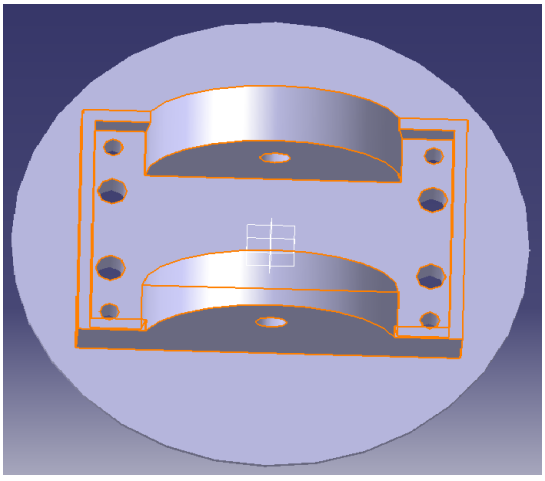


Figura 3-39. Pieza Unión Flexible 2

3.3.2 Construcción Brazo Compliant.

A continuación, se realizará una breve explicación de como se deben de montar las articulaciones flexibles. En función de la longitud que se desee para el robot, se pueden poner más o menos uniones. En este caso, se ha decidido dejar tres articulaciones que unen cuatro actuadores.

1. El primer paso será unir dos bases a los dos extremos de los robots. Para ello se usará una unión fija al motor y otra móvil. Se cogerán con cuatro tornillos penetrantes para que no choquen con las piezas de unión.
2. Conectar a cada una de las bases la unión correspondiente usando los cuatro agujeros de las esquinas.[9]
3. Colocar los cuatro muelles en los orificios de una de las partes. Hacer que coincidan con las de la otra pieza y unir ambas dos y alinear los agujeros de sus circunferencias para poder pasar un tornillo dajarlas fijas.
4. Una vez ensamblada las partes, se une a los motores. Repetir el proceso cuantas veces se necesite.

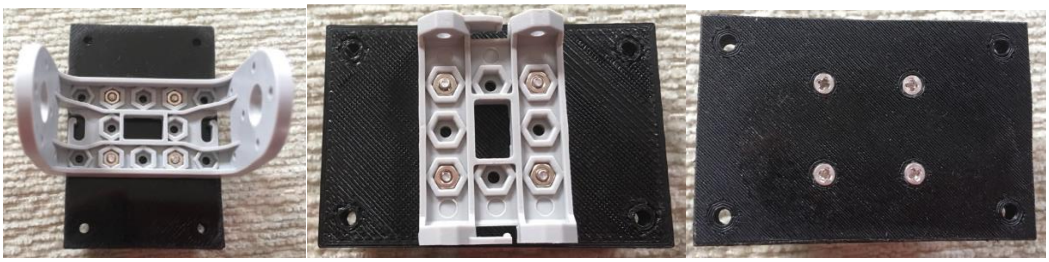


Figura 3-40. Montaje Robot Compliant Paso 1

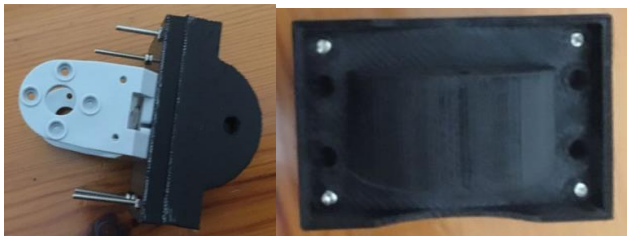


Figura 3-41. Montaje Robot Compliant Paso 2

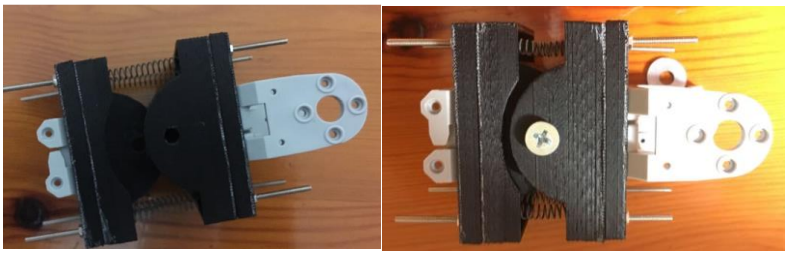


Figura 3-42. Montaje Robot Compliant Paso 3

Figura 3-43. Montaje Robot Compliant Paso 4

[9] Se han usado en este caso tornillos más grandes de los necesario, pero solo para hacer las fotos de la memoria. Estos tornillos se sustituyen por otros de menor longitud.

3.3.3 Cinemática del Robot Compliant:

Para el estudio de la cinemática de este robot se usará el esquema de la figura 3-44, en la cual se pueden observar los grados de libertad $\theta \in \mathbb{R}^4$, que se corresponderán con las articulaciones modificables, es decir, con los motores del robot, así como las articulaciones pasivas, denominadas $\delta \in \mathbb{R}^3$.

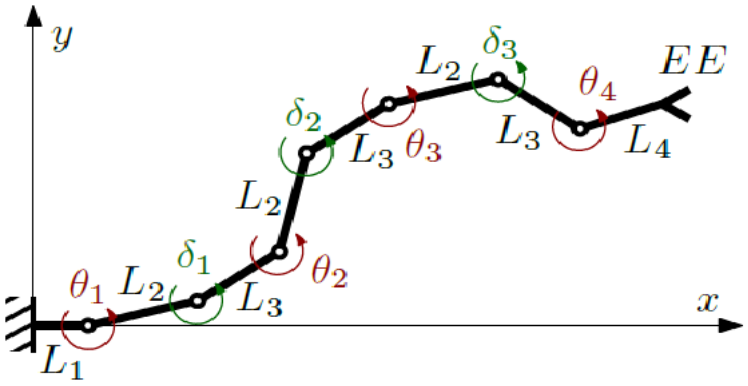


Figura 3-44. Esquema de los Grados de Libertad del Robot Compliant

Se puede expresar la posición y orientación del efector mediante la siguiente expresión:

$$x = L_1\begin{pmatrix}1\\0\\0\end{pmatrix} + L_2\begin{pmatrix}\cos(\theta_1) + \cos(\theta_1 + \delta_1 + \theta_2) + \cos(\theta_1 + \delta_1 + \theta_2 + \delta_2 + \theta_3)\\ \sin(\theta_1) + \cos(\theta_1 + \delta_1 + \theta_2) + \sin(\theta_1 + \delta_1 + \theta_2 + \delta_2 + \theta_3)\\ 0\end{pmatrix}$$
$$+ L_3\begin{pmatrix}\cos(\theta_1 + \delta_1) + \cos(\theta_1 + \delta_1 + \theta_2 + \delta_2) + \cos(\theta_1 + \delta_1 + \theta_2 + \delta_2 + \theta_3 + \delta_3)\\ \sin(\theta_1 + \delta_1) + \cos(\theta_1 + \delta_1 + \theta_2 + \delta_2) + \sin(\theta_1 + \delta_1 + \theta_2 + \delta_2 + \theta_3 + \delta_3)\\ 0\end{pmatrix}$$
$$+ L_4\begin{pmatrix}\cos(\theta_1 + \delta_1 + \theta_2 + \delta_2 + \theta_3 + \delta_3 + \theta_4)\\ \sin(\theta_1 + \delta_1 + \theta_2 + \delta_2 + \theta_3 + \delta_3 + \theta_4)\\ 0\end{pmatrix} + \begin{pmatrix}0\\0\\ \theta_1 + \delta_1 + \theta_2 + \delta_2 + \theta_3 + \delta_3 + \theta_4\end{pmatrix}$$

Ecuación 3-10. Cinemática Directa del Robot Compliant

Con este cálculo, se puede expresar la velocidad del efector usando el jacobiano como

$$v = \dot{x} = J_\theta\dot{\theta} + J_\delta\dot{\delta}$$

Ecuación 3-11. Velocidad Final del Efector

Con los jacobianos definidos como $J_\theta \in \mathbb{R}^{3\,x\,4} =: \frac{\delta x}{\delta\theta}$ y $J_\delta \in \mathbb{R}^{3\,x\,3} =: \frac{\delta x}{\delta\delta}$.

3.4 Sistema de Visión.

3.4.1 Base Matemática para la Obtención de los Ángulos.

El método empleado para la adquisición de los ángulos se basa en la triangulación. Para ello se unirán los centros de dos motores consecutivos junto con el centro que corresponde a la articulación que los une (Figura 6-2), por lo que se tendrán tres puntos que nos permitirán definir un triángulo.

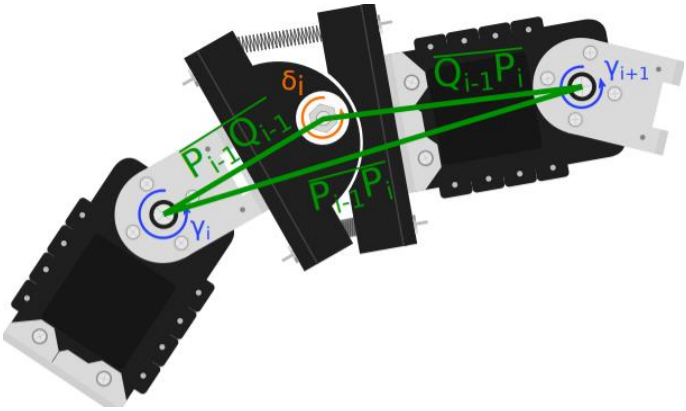
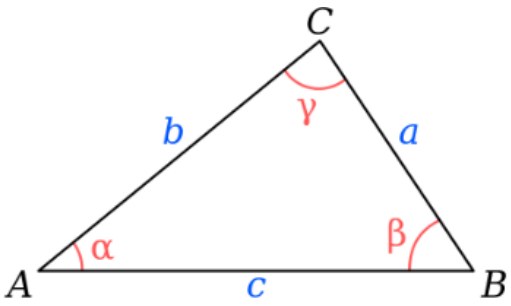


Figura 3-45. Triangulación Motor - Articulación - Motor

El ángulo que se necesita para el cálculo del efector es el correspondiente a la articulación flexible. Por tanto, una vez obtenido el triángulo, se calculará la distancia que separa a cada uno de los vertices y así poder usar el teorema del coseno:



$$c^2 = a^2 + b^2 + 2ab * \cos\gamma$$

Figura 3-46. Teorema Del Coseno

Como se disponen de las longitudes de los lados del triángulo, se puede despejar el ángulo γ y así obtener la medida necesaria.

Por tanto, se puede obtener directamente la medida del ángulo necesario, obteniendo la ecuación 3-12.

$$\gamma = \cos^{-1}\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$$

Ecuación 3-12. Ángulo Gamma

3.4.2 Procesamiento de Imagen.

Una vez descrito el proceso a seguir para la medición de los ángulos, se plantea el procedimiento para obtener, de la manera más precisa posible, los puntos que se corresponden con los centross de cada una de las articulaciones y de los giros de los motores.

Para ello lo primero que se hace es numerar cada uno de los centros a hallar del 1 al 7 y se asocia un color. Siguiendo los colores que se han decidido poner para el robot, en este caso concreto será:

Una vez definidos, se deberá comenzar a leer la imagen de video. Se necesitarán 7 máscaras diferentes, una por cada color. Con esto creará una matriz binaria, en la cual los píxeles que se correspondan con el valor 1 se corresponde con el color correspondiente, y 0 con el resto de los colores.

Para que la máscara quede bien definida y se evite, en medida de lo possible, el ruido. Se le aplicará a cada una de ellas las operaciones morfológicas de cierre y apertura. Con la operación de cierre se conseguirá que los posibles grupos de píxeles suficientemente pequeños como para considerarlos que no corresponde con la parte de la imagen que se quiere aislar.

Por contra, la operación de apertura conseguirá que los pixels que deberían de haberse reconocido (por estar rodeados de píxeles que si lo han sido), se consigan reconocer. Gracias a ello, se obtendrá una imagen nítida y que no tendrá apenas ruido, por lo que se logrará tener un circulo bastante preciso.

Una vez se haya logrado filtrar la imagen, se usarán los momentos, los cuales nos dan información geométrica de la region plana. Estos momentos se calculan como:

$$m_{pq} = \sum_{x0}^{N-1} \sum_{y0}^{M-1} x^p y^p F(x,y) \qquad con \qquad F(x,y) \rightarrow Imagen \qquad p,q = 0,1,2 \dots$$

Ecuación 3-13. Momentos Geométricos

En este caso, se usarán los siguientes momentos:

- Momento de orden Cero: p = q = 0. → Coincide con el área del objeto descrito.
- Momentos de orden Uno: (p = 0, q = 1) y (p = 1, q = 0) → Permiten obtener el centro de gravedad.

Por tanto, para cada una de las máscaras se calculará el momento cero. Si este momento es mayor a un determinado valor, significará que está reconociendo el objeto y se le calculará su centro de gravedad realizando la operación:

$$x = \frac{m_{10}}{m_{00}} \qquad y = \frac{m_{01}}{m_{00}}$$

Ecuación 3-14. Centro de Gravedad

Gracias a este procedimiento, se obtendrán los 7 centros de los 7 colores, teniendo así los valores x e y de cada uno de ellos y pudiendo realizar los cálculos matemáticos expuestos en el apartado anterior.

4 CONTROL Y SIMULACIÓN.

"Elige un trabajo que te guste y no tendrás que trabajar ni un día de tu vida"

- Confucio -

Proseguiremos con la explicación del control que se realizará para cada uno de los brazos de robots. Para ello nos centraremos por separado en cada uno de los brazos y se explicará el control aplicado. Para el control del robot industrial, se ha realizado una simulación en Matlab que permitirá comprobar que el controlador funciona como debería.

4.1 Introducción.

Se persigue como objetivo principal poder probar el control para el Robot Compliant. Al tener una configuración con articulaciones pasivas, como son las articulaciones con amortiguación, en las cuales no podemos ejercer ningún tipo de acción, se decide primero realizar el control del brazo robótico industrial ya que es más simple, permitiendo así crear una base sólida sobre la que construir el control del Compliant.

Por tanto, se comenzará explicando las bases del control del brazo de robot industrial. Será un control en velocidad que usará el jacobiano para definir las velocidades articulares que deberá tener cada uno de los grados de libertad del robot.

Una vez quede este control con las bases sentadas, será cuestión de añadir modificaciones para el robot Compliant, pero partiendo desde el controlador ya implementado en el robot industrial.

En los siguientes subapartados se podrá comprobar el estudio teórico de los controladores. La programación discreta del mismo se podrá ver en el siguiente capítulo.

4.2 Control Brazo Robótico Industrial 6 GDL.

Se decide implementar un control en velocidad para el robot, por tanto, se tendrá que calcular en cada instante la velocidad que es necesaria aplicar en cada uno de los grados de libertad del brazo para que el extremo llegue a la posición deseada con una determinada orientación. Para ello se usará el anteriormente explicado jacobiano de velocidades, al cual se le introducirán una serie de modificaciones para evitar configuraciones que puedan provocar singularidades y afecten de forma desfavorable al robot.

Se comenzará usando como datos la posición y orientación de destino, el error que existe entre el destino y la posición actual, que se obtiene con la posición instantánea de cada una de las articulaciones. Por tanto, los datos de nuestras ecuaciones serán:

$$x_d = pos\ deseada = [x_d \quad y_d \quad z_d \quad \alpha_d \quad \beta_d \quad \gamma_d]^T \leftrightarrow \phi(q_d) = [q_{d1} \quad q_{d2} \quad q_{d3} \quad q_{d4} \quad q_{d5} \quad q_{d6}]^T$$

$$x = pos\ actual = [x \quad y \quad z \quad \alpha \quad \beta \quad \gamma]^T \quad \leftrightarrow \quad \phi(q) = [q_1 \quad q_2 \quad q_3 \quad q_4 \quad q_5 \quad q_6]^T$$

$$error\ posción = e = x_d - x = \phi(q_d) - \phi(q) \quad \leftrightarrow \quad error\ en\ velocidad = \dot{e} = \dot{x}_d - \dot{x}$$

Ecuación 4-1. Datos de Control

Como resulta natural, la velocidad en el punto objetivo será nulo, por tanto:

$$Jacobiano\ de\ posiciones = J(q) \qquad velocidades\ articulares = \dot{q}$$

$$\dot{x}_d = v_d = 0 \qquad \leftrightarrow \qquad \dot{e} = -\dot{x} = -J(q) \cdot \dot{q}$$

Ecuación 4-2. Ecuación de error en Velocidad

A continuación, nos centraremos en la expresión J^{-1}, la inversa del jacobiano. En el caso que nos ocupa, el jacobiano resulta ser una matriz 6x6, lo que implica que es cuadrada y tiene inversa. Aun así, se le realizará un procedimiento para que la matríz sea invertible. Tambíen se le aplicará el anteriormente comentado procedimiento para evitar singularidades. Como resultado, se obtendrá la matriz $J^{\dagger}$.

$$J(q)^{\dagger} = J(q)^T \cdot \left(J(q) \cdot J(q)^T + k_{jac}^2 \cdot I\right)^{-1} \quad con$$

$$k_{jac} \coloneqq k_0 \cdot exp\left(-\frac{\det(J \cdot J^T)}{2 \cdot \varepsilon^2}\right) \quad , \qquad k_0 > 0, \qquad \varepsilon \ \epsilon \ (0,1]$$

Ecuación 4-3. Jacobiano Inverso sin Configuraciones Singulares.

El valor k_{jac} irá multiplicado por la matriz identidad, y servirá para limitar las velocidades de las articulaciones. Conforme un grado de libertad se acerque a una configuración de singularidad, k_{jac} irá haciéndose cada vez mayor, por lo que al estar elevado a -1, empequeñecerá el valor de $J(q)^{\dagger}$ para dicha articulación, y por tanto la actuación que se ejercerá sobre ella irá volviéndose cada vez menor, evitando así que llegue a configuraciones no deseadas.

Una vez hallado el jacobiano inverso, hay que definir la función de control que se implementará. Se implementará un control PI.

$$señal\ de\ control = k_p \cdot e(t) + k_i \int_o^t e(\tau)\, d\tau$$

Ecuación 4-4. Controlador PI en el tiempo

Por tanto, en la ecuación 4-4 se debe sustituir las variables de control, obteniendo:

$$k_p \cdot e + k_i \cdot \zeta \qquad con\ \ \zeta = \zeta_{anterior} + \Delta T \cdot e, \quad \zeta \in [-\zeta_{min}, \zeta_{max}] \rightarrow Anti\ WinUp$$

Ecuación 4-5. Control PI Discreto

Para finalizar el control, hay que concretar la variable del error, ya que se le calculará su error cuadrático medio, a su vez dividido por una constante. Esto permitirá que cuando el robot se encuentre en una posición muy alejada de la posición deseada, no aplique una gran velocidad desde el comienzo del movimiento, sino que sea algo progresivo.

$$e = \frac{x_d - x}{1 + 0.5 \cdot (x_d - x)^2}$$

Ecuación 4-6. Error Cuadrático Medio.

Si se aplican todos los cambios realizados a la ecuación 4-6, se obtiene una ecuación en la cual obtendremos las velocidades articulares, haciendo uso del nuevo jacobiano que limita las singularidades y un control PI que hace uso del error cuadrático medio para evitar altas velocidades cuando se encuentra alejado del punto de destino.

$$\dot{q} = J(q)^{\dagger} \cdot \left(k_p \cdot e + k_i \cdot \zeta\right)$$

Ecuación 4-7. Ecuación de Control Final.

4.3 Simulaciones Robot Industrial 6 GDL:

4.3.1 Comprobación Cinemática:

Para la solución de este problema cinemático directo, se han creado varios archivos .m en Matlab, que permiten definir los parámetros del robot y nos proporcionan los resultados deseados. Estos archivos son:

- CinematicaDirectaSym.m → Si se ejecuta, se obtendrán las matrices de transformación individulales de cada una de las articulaciones, así como la matriz T, todo calculado en simbólico.
- ComprobacionCinematicaRad.m → Nos permite conocer las matrices de transformación para un caso concreto. Se podrá modificar los grados que están girados cada articulación (en radianes).
- ComprobacionCinematicaDeg.m → Caso idéntico al anterior, pero con las coordenadas articulares en grados.
- CinematicaTool.m → Hace uso del Toolbox de Peter Corke. Permite crear una simulación del robot e ir modificando los valores de las articulaciones para ver como quedaría el brazo en el espacio de trabajo. Tambíen proporciona el punto final del extremo, así como su orientación.

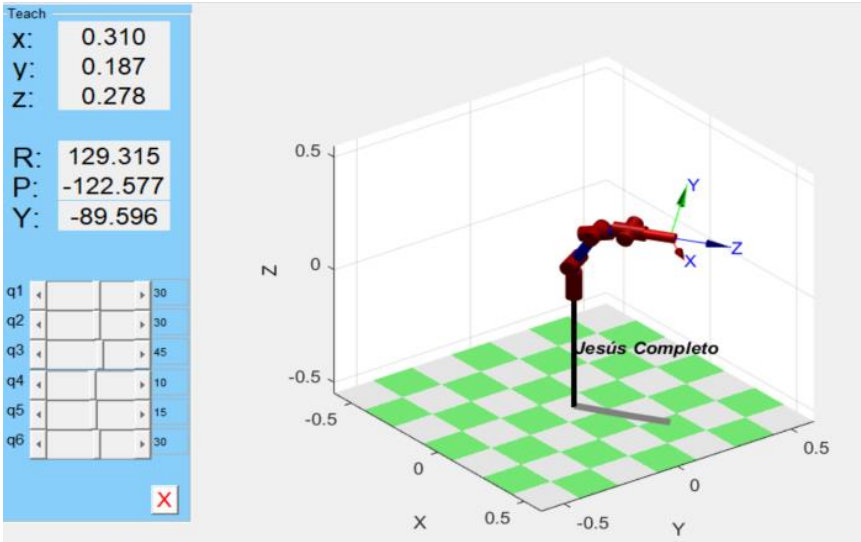


Figura 4-1. Simulación Robot Industrial con ToolBox de Peter Corke.

Se dispone de otro archivo.m llamado Inversa.m que permitirá poner un punto del espacio cartesiano y su orientación, y nos dará como resultado las posiciones que deberán tener los grados de libertad del robot para conseguir que el extremo cumpla los datos introducidos.

Por tanto, se ha realizado comprobaciones para asegurar que los cálculos de las cinematicas sean correctos. El procedimiento ha sido:

- Se busca un punto alcanzable por el robot con el archivo CinematicaTool.m.
- Se colocan los grados deseados en el archivo CinDirGrados/CinDirRadianes.m y se comprueba que se llega al mismo punto con la misma orientación (eso implica que los cálculos en simbólico están bien realizados, ya que es el mismo procedimiento para el cálculo).
- Se colocan los parámetros de posición y orientación en el archivo Inversa.m obteniéndose los valores de las articulaciones. Se comprueba si se corresponden con los que debería tener.

4.3.2 Simulación del Control:

Se dispone de un archivo matlab denominado ControlPIKOvariable.m en el cuál se ha implementado el control que se va a realizar. El experiment concreto cuenta con un número de 500 muestras (cada una correspondiente a 0.1 segundos). El experimento constará de ir de un punto a otro y observar como irán modificando las velocidades y posiciones del robot hasta llegar a su destino, comprobando que el error en regimen permanente sea nulo.

Se legará al punto →	x = 0.065 (m)	y = 0.284 (m)	z = 0.132 (m)
Con la orientación →	alpha = 0 (rad)	betta = 0.698 (rad)	gamma = 0 (rad)

Se partirá con los siguientes valores de coordenadas articulares:

$$q_1 = 10º \quad q_2 = 30º \quad q_3 = 30º \quad q_4 = 45º \quad q_5 = 30º \quad q_6 = 60º$$

Una vez ejecutado el script, se dispone de otro llamado plotstrayectorias.m que al ejecutarlo nos mostrará las graficas con los resultados obtenidos. Podemos observarlas en las siguientes figuras:

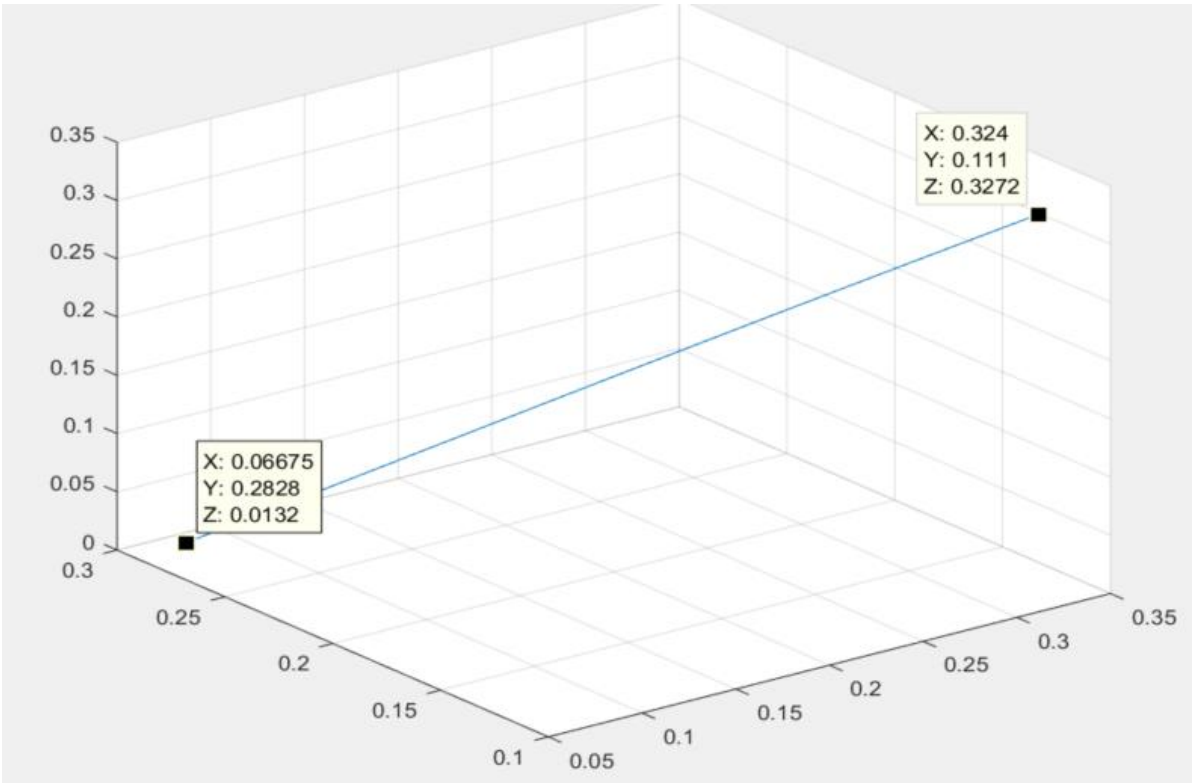


Figura 4-2. Trayectoria Robot Industrial.

I. Gráficas de Posiciones Articulares:

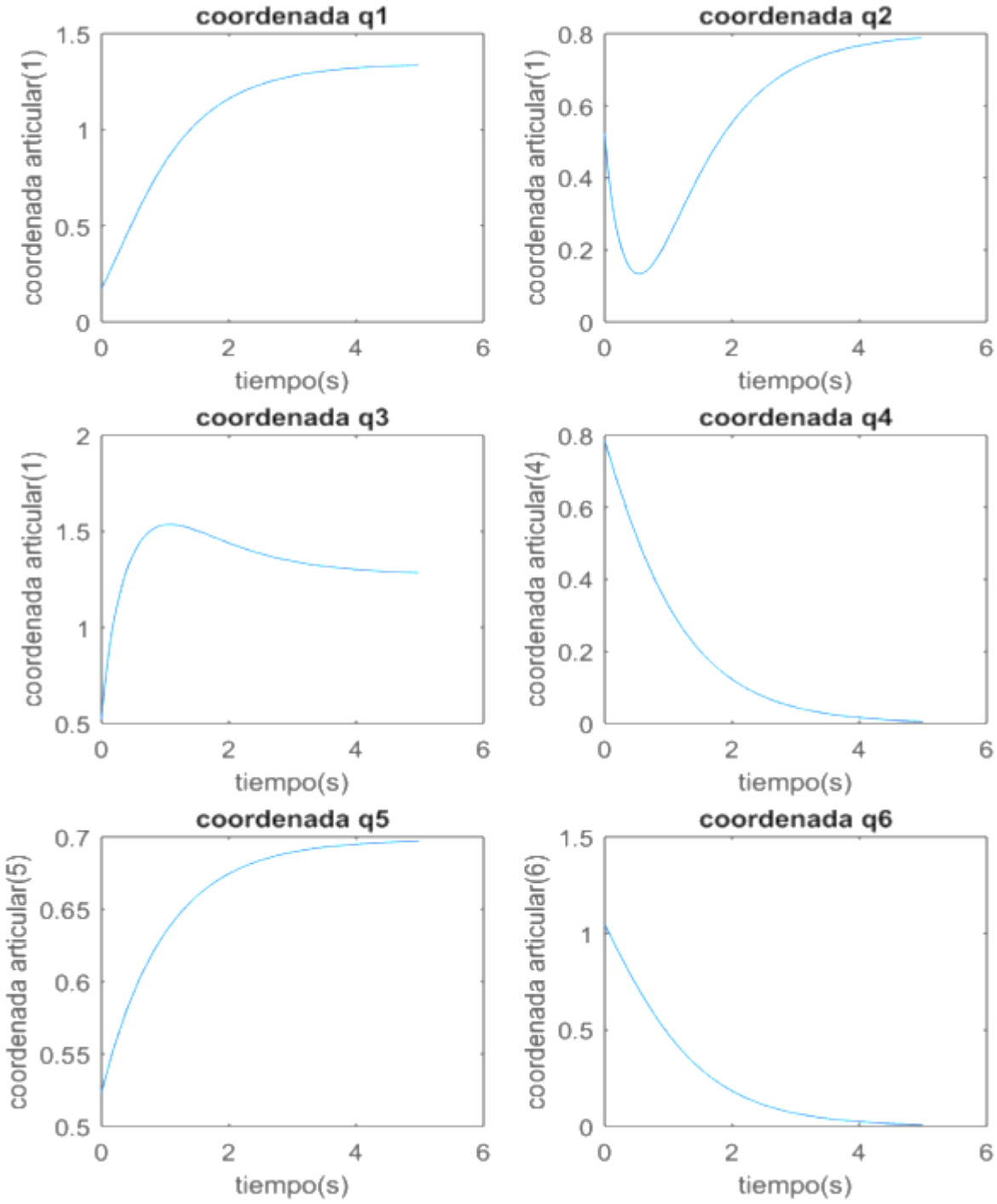


Figura 4-3. Trayectoria Articular

Se puede observar como cada uno de los grados de libertad consigue llegar a la posición adecuada para que el extremo esté con las condiciones deseadas de posición y orientación.

II. Gráficas de Velocidades Articulares:

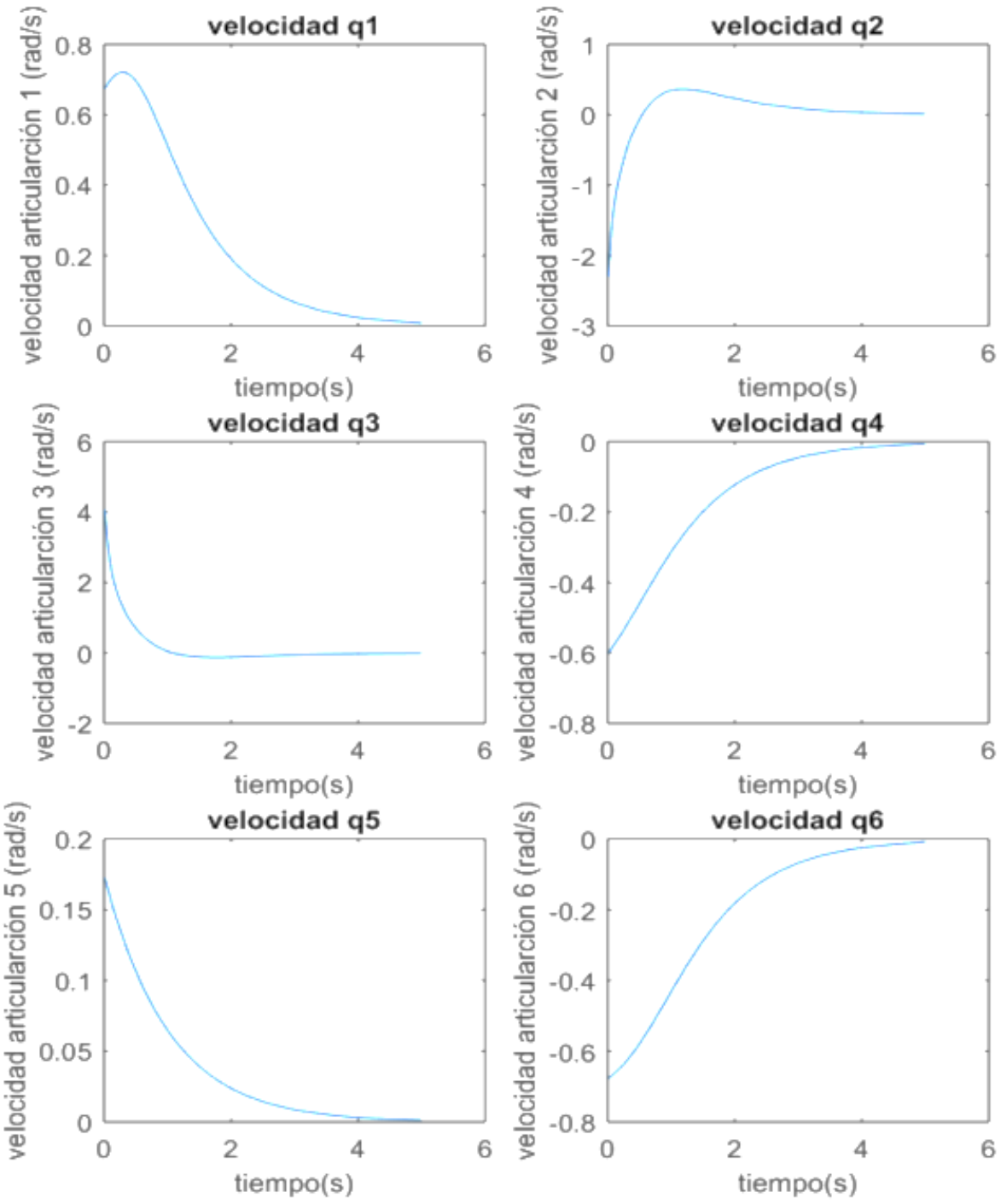


Figura 4-4. Velocidad Articular

En esta ocasión se ven las velocidades que han debido aplicarse a cada una de las articulaciones para cumplir el objetivo.

III. Gráficas de Errores:

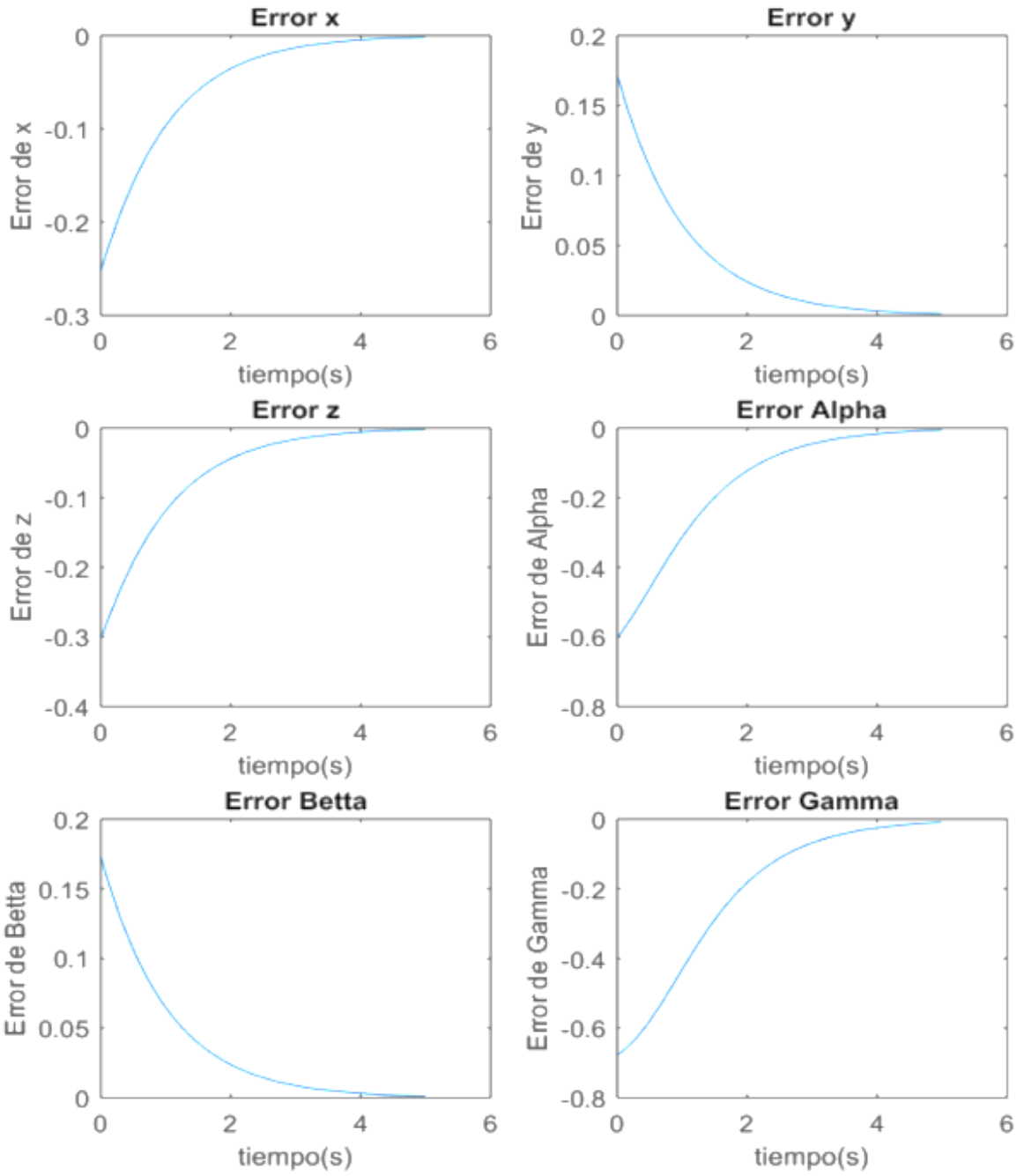


Figura 4-5. Errores Posición y Orientación

Se finaliza el experimento con las gráficas de los errores en posición y orientación, los cuales van disminuyendo conforme el control va avanzando y consigue llegar al destino. Vemos como el error es aproximadamente nulo.

5 EXPERIMENTACIÓN Y ANÁLISIS DE RESULTADOS.

"El paso más importante que puede dar una persona será siempre el siguiente"

- Brandom Sanderson -

Llegados a este punto en el que se tienene los dos brazos de robots contruidos y listos para su uso, hay que centrarse en la configuración experimental. Recordamos al lector que el objetivo de este proyecto se centra en poder sentar las bases para poder probar el control del robot compliant.

Por tanto, a lo largo de este capítulo, podrá comprobarse como se ha preparado el experimento, así como la programación realizada tanto en la RaspBerry Pi como en el Arbotix – M.

5.1 Introducción.

El experimento que se desea realizar trata de hacer que el robot industrial sea capaz de coger un objeto, como puede ser una pelota, de un punto prefijado del espacio de trabajo del robot. Una vez recogido, el brazo lo colocará cerca del robot Compliant. Cuando el robot Compliant reciba el estímulo de que el objeto está dispuesto en su podición, comenzará su movimiento.

El robot compliant comenará en una posición incial recogida cerca de la zona interior de la region de manipulabilidad. Cuando se le indique, el robot se desplegará y comenzará su interactuación con el robot industrial. El Compliant tratará de ejercer presión sobre la pelota que estará sujetando el robot industrial, lo que provocará la deflexión de las articulaciones flexibles, lo que implicará que el control tenga que recalcular las actuaciones de los motores para que siga ejerciendo una presión sobre el objetivo.

Se puede ver un esquema del experimento en la siguiente figura:

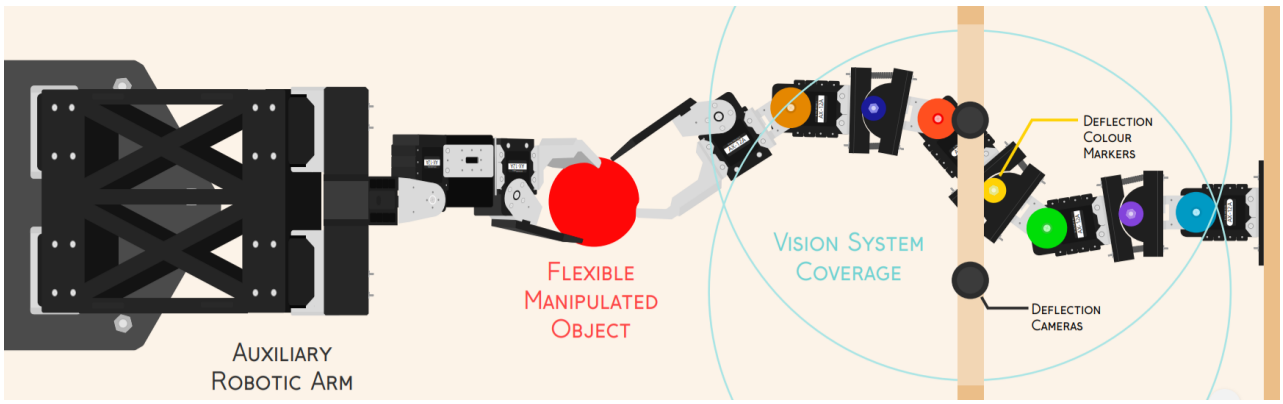


Figura 5-1. Experimento

5.2 Configuración de Visión Artificial para Robot Compliant.

Como se ha podido observar, uno de los principales "inconvenientes" que posee el robot Compliant es que tiene articulaciones pasivas, en las cuales no tenemos actuadores, por lo que tampoco se tiene forma de recabar de manera directa información de ellas.

Surge la necesidad de diseñar un sistema que sea compatible con resto del conjunto y que realice la función de extraer la información necesaria de los ángulos que forman las articulaciones flexibles.

Para ello se recurre a la RaspBerry Pi y a OpenCV, que permitirán realizar un programa en Python capaz de realizar dicho cometido. A continuación, se expondrá el procedimiento seguido para cumplir dicho objetivo.

5.2.1 Función de Visión y Método Empleado.

La función principal de usar la RaspBerry es poder implementar un sistema de visión artificial que sirva de apoyo al brazo robótico compliant, realizando la medición de los ángulos que forman las articulaciones flexibles.

Para ello se usará una webcam que capturará videos en tiempo real y realizará una serie de procesos que permitan obtener los centros de giro tanto de los motores como de las articulaciones los cuales estarán pintados de diferentes colores, lo que permitirá asociar cada color con una de las articulaciones o motores, permitiendo así realizar los cálculos necesarios para la obtención de los ángulos, además de facilitar en gran medida el código que procesará las imágenes de video.

En la siguiente figura se observa el robot completo con los centros pintados cada uno de un color.

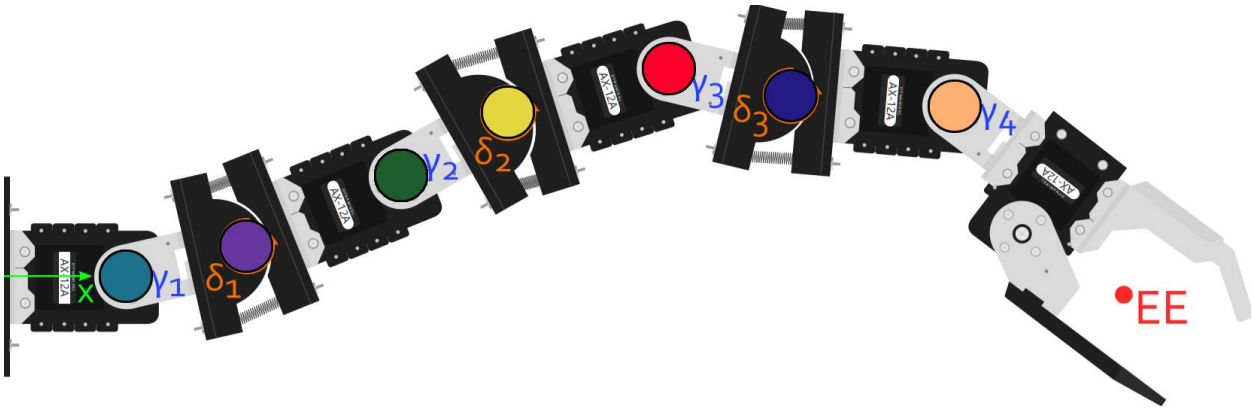


Figura 5-2. Brazo Robot Compliant

Centro Número 1	Motor	Cyan
Centro Número 2	Articulación flexible	Morado
Centro Número 3	Motor	Verde
Centro Número 4	Articulación flexible	Amarillo
Centro Número 5	Motor	Rojo
Centro Número 6	Articulación flexible	Azul
Centro Número 6	Motor	Crema

Tabla 5-1. Tabla de Centros

5.2.2 Pseudo código Implementado en la RaspBerry Pi.

Se disponen de dos archivos.py que serán los encargados de obtener la medida de los ángulos aplicando el método descrito en el punto anterior.

El primer ejecutable denominado Frames creará una imagen de video en la cuál se obtendrá las imágenes capturadas por la webcam, junto con unas barras de desplazamiento que permitirán ajustar los parámetros de la imagen en HSV para poder aislar el color que se desee.

En el otro ejecutable se deberán insertar los valores obtenidos de asilar los colores con la imagen anterior. Una vez hecho esto, cuando se ejecture imprimirá por pantalla los datos de los ángulos medidos, así como dos imágenes de video en las cuales se podrán ver los centros y demás datos de interés.

- **Cabeceras:**
  - Bibliotecas.
  - Variables usadas. Es necesario ajustar en este punto los márgenes para el reconocimiento de imagen. Para ello hay que ejecutar antes un ejecutable que permitirá realizar este proceso.
- **Comenzar a grabar la imagen.**
- **Convertirla a HSV.**
- **While (1):**
  - Espera 5 segundos para que se estabilice la imagen.
  - Filtrar el ruido con operaciones morfológicas.
  - Encontrar momento central para cada máscara → Área de cada máscara.
  - Si Área de máscara > NumPixeles → Modificar centro del color adecuado.
  - Calcular distancias entre los centros adecuados:
    - Triángulo 1 → Centros 1, 2 y 3 → Calcular Gamma1
    - Triángulo 2 → Centros 3, 4 y 5 → Calcular Gamma2
    - Triángulo 3 → Centros 5, 6 y 7 → Calcular Gamma3
  - Imprimir por pantalla imágenes con los resultados.
  - Imprimir por puerto serie los resultados
- **Fin While (1).**

5.3 Programación Arbotix – M.

Como ya se ha comentado con anteriorirdad, se desea realizar un control en velocidad para los brazos de robots, pero los motores no tienen la posibilidad de controlarlos de esta manera, ya que el modo "rueda" o de giro continuo que posee modifica el torque de los motores, no la velocidad.

Por consiguiente, para alcanzar el objetivo del control en velocidad se ha tenido que usar el modo servo motor, en el cuál si se puede especificar la velocidad a la que se dese ir en cada grado de libertad, aunque para ello, se ha tenido que calcular previamente una posición de destino.

Para conseguir realizar esto, se ha buscado experimentalmente un tiempo adecuado durante el cuál se mantendrá la velocidad constante. En el caso del robot industrial, esta velocidad ha sido 3 milisegundos. De esta manera, sabiendo la posición actual, la velocidad definida y el tiempo durante el que se mantendrá, se puede calculary la velocidad de destino.

A continuación se expondrá un pseudocódigo en el que se podrá ver el proceso seguido para implementar el control en el microcontrolador.[10]

5.3.1 Pseudo Código implemrntado en Arbotix – M.

- **Cabeceras:**
  - Bibliotecas.
  - Variables usadas.

- **Void Setup:**
  - Reserva de memoria para puerto serie.
  - Activación puerto serie.
  - Habilitación de torque de motores.

- **Void Loop:**
  - Reset Flags.
  - Leer posiciones.
  - Calcular posición y orientación actual.
  - Elige punto de destino.
  - Cálculo del error

[10] En el Anexo del documento se podrá ver el código implementado.

- Hacer:
    - Calcular Jacobiano.
    - Calcular Dumpping.
    - Calcular Pseudo Iinversa Jacobiano con Dumping
    - Calcular velocidad de actuación.
    - Calcular posición de destino.
    - Calcular actuación de los motores.
    - Leer posiciones.
    - Calcular posición actual.
    - Calcular error.
    - Si iteración < NumIteraciones → iteración ++
    - Si no → OverFlow (No converge la posición); Imprime por pantalla error.
- Mientras que Flag = 0.
- Siguiente punto de destino.

- **Fin:**

5.4 Estructura del Banco de Pruebas.

Surge la necesidad de tener un espacio de trabajo bien definido para la realización de los experimentos, en el que se pueda colocar la cámara para realizar la vision y tenga buena iluminación, donde esté colocados y anclados los dos robots, así como todas las conexiones que deben de realizarse para la comunicación de ambos robots con el Arbotix – M y la RaspBerry Pi.

Para realizarlo se ha usado un tablero de madera de 120 x 60 centímetros, al cuál se le ha añadido dos vigas de madera verticales y una horizontal para colocar la cámara a la altura necesaria. En las escuadras de las vigas se han añadido dos focos de leds de luz blanca para iluminar la zona de trabajo del robot compliant, así podrá tener la suficiente luminosidad para que la cámara pueda obtener los datos de manera adecuada.

Una vez realizado eso, se decidió la posición que ocuparán cada uno de los robots durante el experimento y se han anclado a la base para que puedan realizar los movimientos de manera correcta.

Concluida la organización de los módulos de mayor tamaño, se procede a la instalación eléctrica y electronica, que conlleva la conexión de las luces a la red elécrtica, la alimentación de la RaspBerry Pi y del Arbotix – M, que a su vez alimentará a los actuadores.

Para realizar este proceso, se colocan en la zona externa al trabajo de los robots una regleta con los suficientes enchufes para conectar los aparatos a la red eléctrica, así como la raspberry y el Arbotix – M, que también serán fijados a la base para evitar que se muevan. Tras esto, se procede a poner canaletas por las que irán los cables, evitando que queden en la zona de trabajo de los robots y puedan provocar problemas a la hora del experimento.

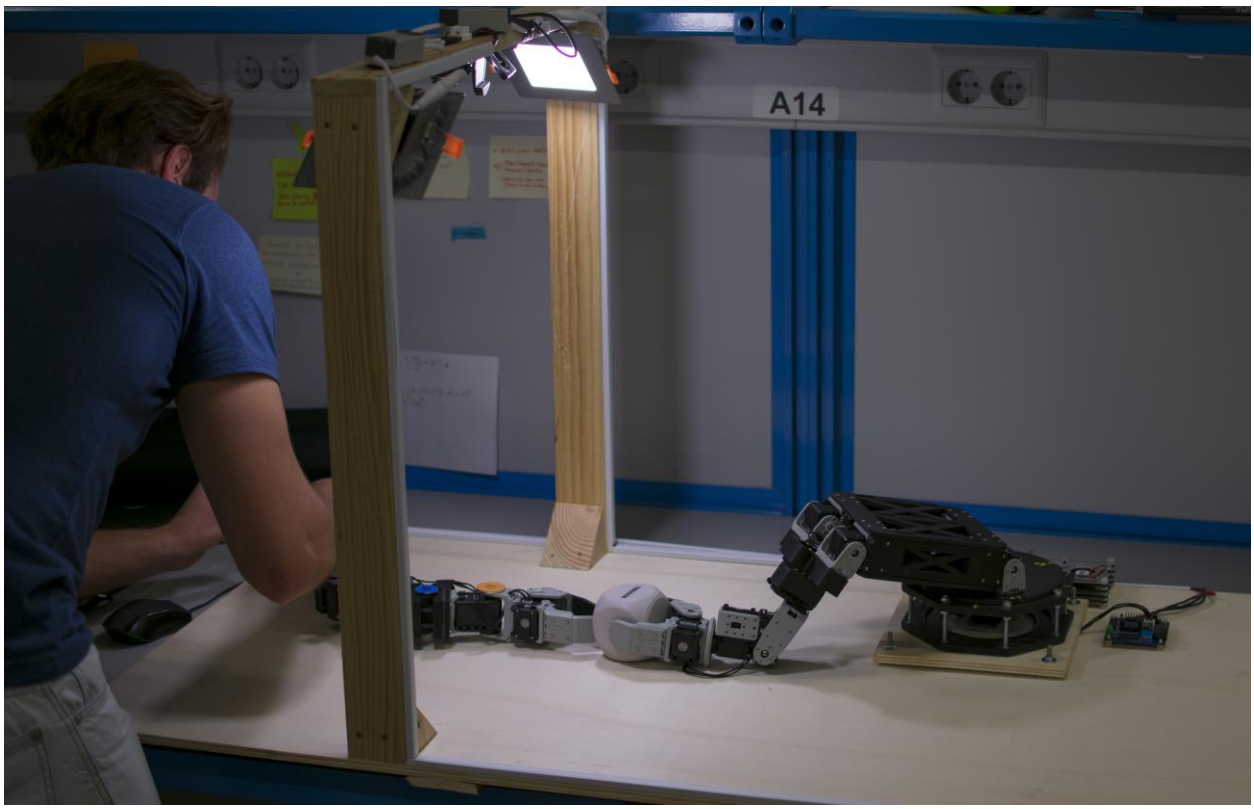


Figura 5-3. Estructura del Banco de Pruebas

5.5 Experimento.

El experimento que se ha realizado consiste en los siguientes pasos:

1. Los robots partirán de una posición de inicio. Se colocará un dado de goma espuma en una posición definida.

Figura 5-4. Posición Inicial

2. El robot manipulador cogerá el dado e irá a una posición intermedia. Tras esto, lo acercará a una posición cercana para el robot compliant.

3. El robot Compliant primero irá a una posición en la cual estará enroscado. Tras esto se desenrollará, y por último irá a la posición en la que toque el dado.

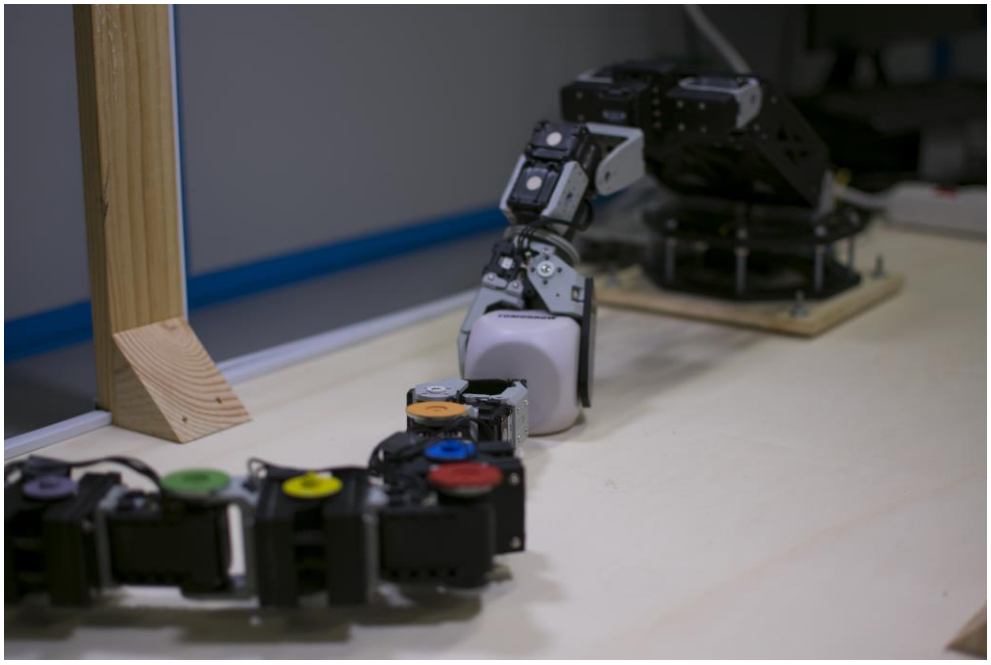


Figura 5-5. Posición Final 1

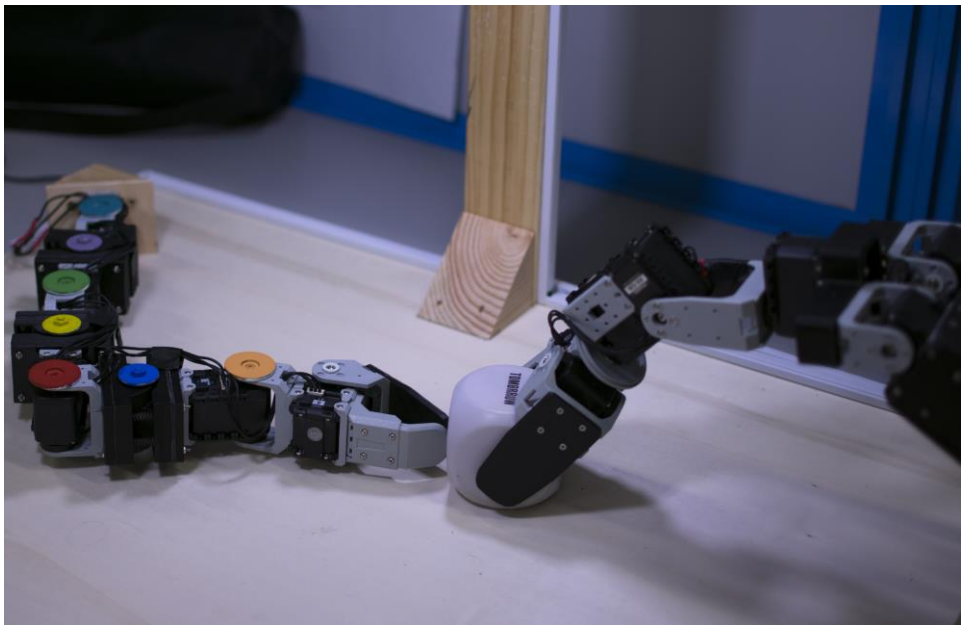


Figura 5-6. Posición Final 2

5.5.1 Objetivo del Experimento:

Los objetivos del experimento son los siguientes:

- Observar el comportamiento del controlador implementado en el robot industrial de 6 grados de libertad, ya que el control que e realizará para el compliant será a partir del ya implementado en este robot.
- Comprobar la precisión de los servo motores y si son capaces de llegar a la posición deseada con un error suficientemente bajo (a poder ser, error nulo).
- Hacer uso de la RaspBerry y la visión artificial y comprobando su eficiencia. Para ello se han puesto puntos alejados a los que deberá moverse el robot compliant, provocando así las deflexiones de los muelles y por tanto, comprobando el funcionamiento de la visión.

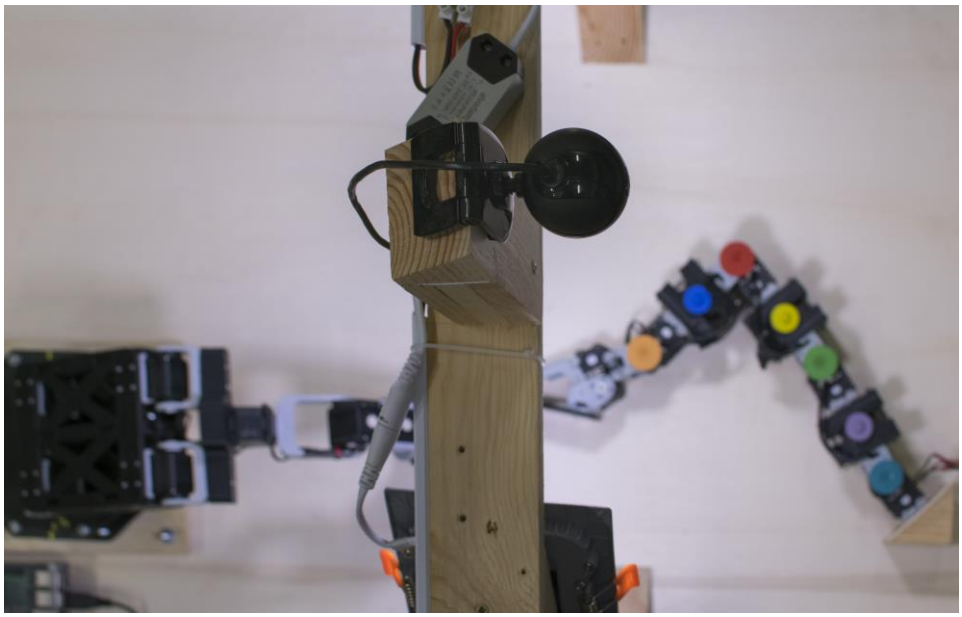


Figura 5-7. Posición final 3

5.5.2 Resultados de la Experimentación.

Tras la realización del experimento, se comprueba que el control aplicado al robot industrial cumple con su cometido, siendo capaz de calcular las velocidades que se deben de aplicar a cada uno de los grados de libertad del robot, así como el funcionamiento de la visión realizada para la realimentación del control compliant.

A continuación se expondrá un gráfico en el cuál se podrá ver como van variando las posiciones articulares durante el experimento, así como la posición de la pinza. Estos datos han sido obtenido a través del puerto serie del Arbotix y son mostrados haciendo uso de Matlab.

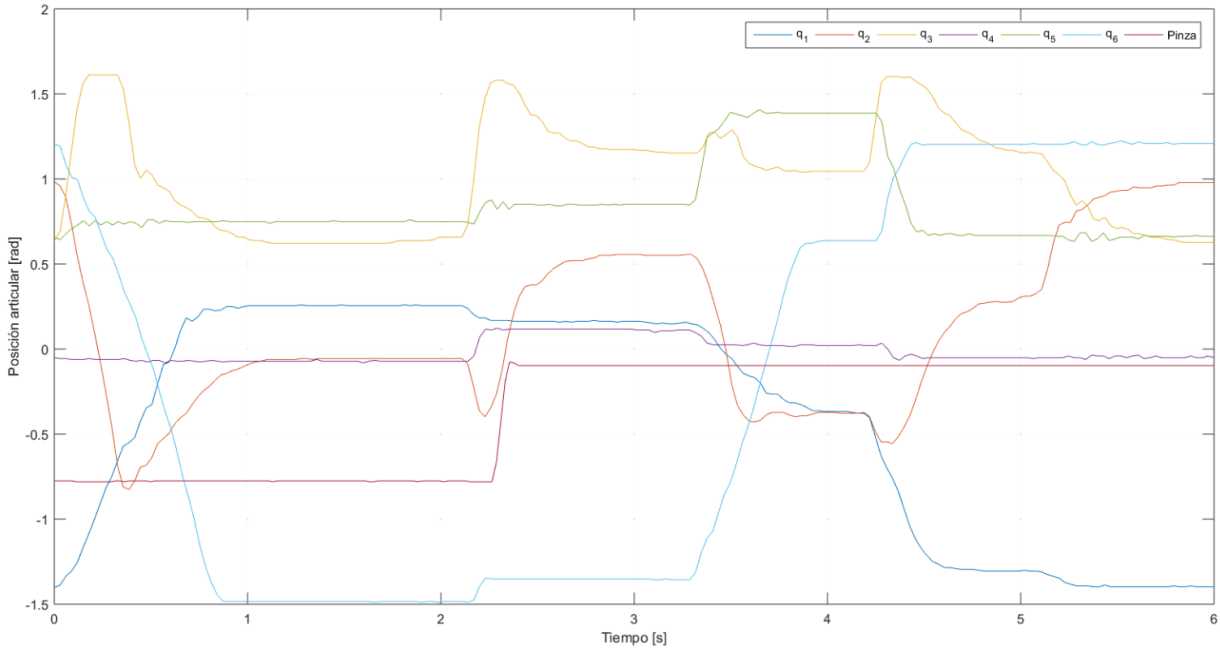


Figura 5-8. Coordenadas Articulares Robot 6 gdl

Se comprueba que el robot ha llegado a la posición de destino haciendo uso de una gráfica en la cuál se puede apreciar como se corrigen los errores en la posición del efector del robot de 6 grados de libertad, convergiendo en cada uno de los puntos y logrando un error prácticamente nulo.

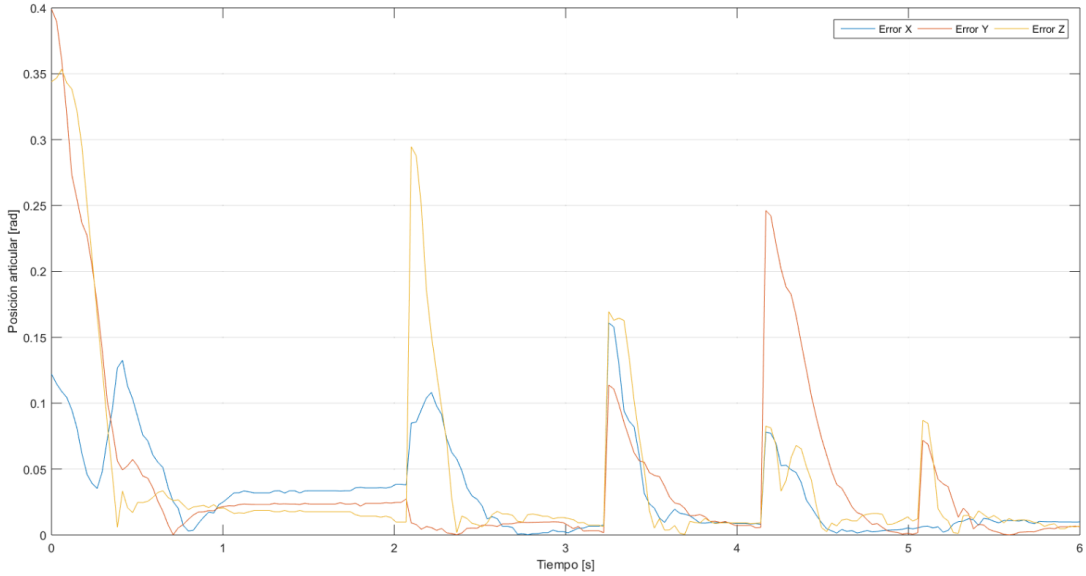


Figura 5-9. Errores Robot 6 gdl

Para el robot compliant se han obtenido las gráficas de cómo se ha producido el movimiento hasta llegar a las posiciones angulares que se definieron para el experimento.

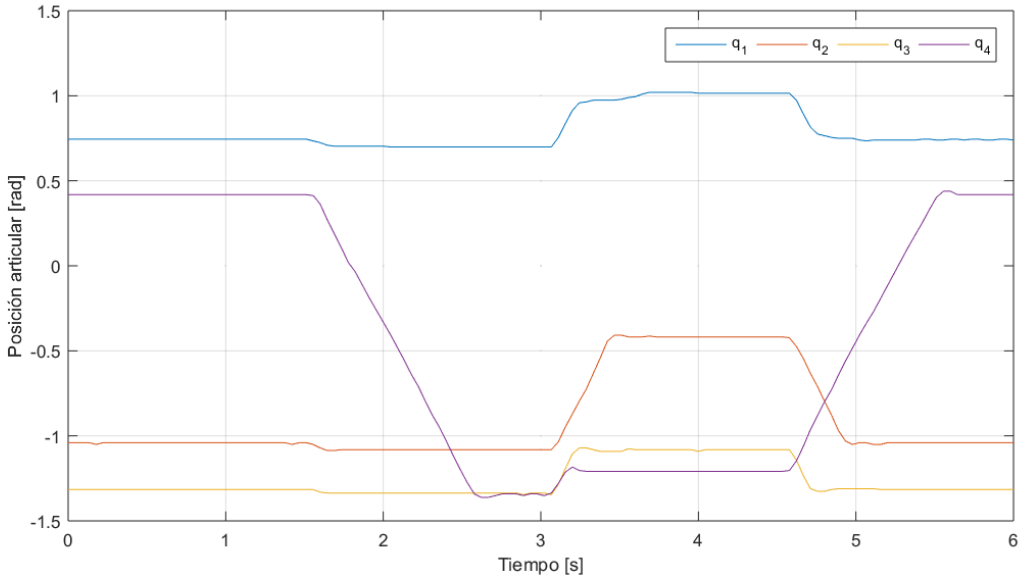


Figura 5-10. Posiciones Articulares Robot Compliant

Para más información, se pueden visitar los siguientes enlaces, en los que se han subido los videos de los experimentos realizados.

Enlace 1: Se podrá ver como los robots han realizado el experimento.

https://www.youtube.com/watch?v=zEN_vbMPtFI&feature=youtu.be

Enlace 2: Visión Artificial.

https://www.youtube.com/watch?v=9yHEv5QEido

6 CONCLUSIONES.

"La verdadera felicidad radica en la finalización del trabajo utilizando tu propio cerebro y tus habilidades"

- Soichiro Honda -

Tras la realizalización de los experimentos y comprobar su funcionamiento, se han llegado a las siguientes concluisiones.

- El objetivo general de crear un sistema robótico de pruebas con dos brazos de robots robustos capaces de experimentar nuevos controladores se ha cumplido de manera muy satisfactoria.
- Para el objetivo específico de diseñar, construir y programar un brazo robótico industrial de 6 grados de libertad se ha desempeñado de forma exitosa, ya que se ha conseguido un prototipo perfectamente funcional, robusto y al cuál se le han programado el control deseado, pudiendo comprobar con los experimentos su perfecto funcionamiento.
- Se ha conseguido realizar el diseño y construcción de un robot flexible con el cuál poder realizar los experimentos deseados para la implementación de nuevos controladores, por lo que el objetivo se ha realizado correctamente.
- El sistema de visión ha sido probado y se comprueba que la RaspBerry Pi carece de suficiente potencia computacional para poder servir de apoyo para el robot compliant. Por tanto, si se desea realizar el control de robots flexibles, se deberá sustituir la la RaspBerry Pi por un hardware de mayor potencia.

7 ANEXOS:

7.1 Anexo I. Listado de components del Kit Bioloid Premium:

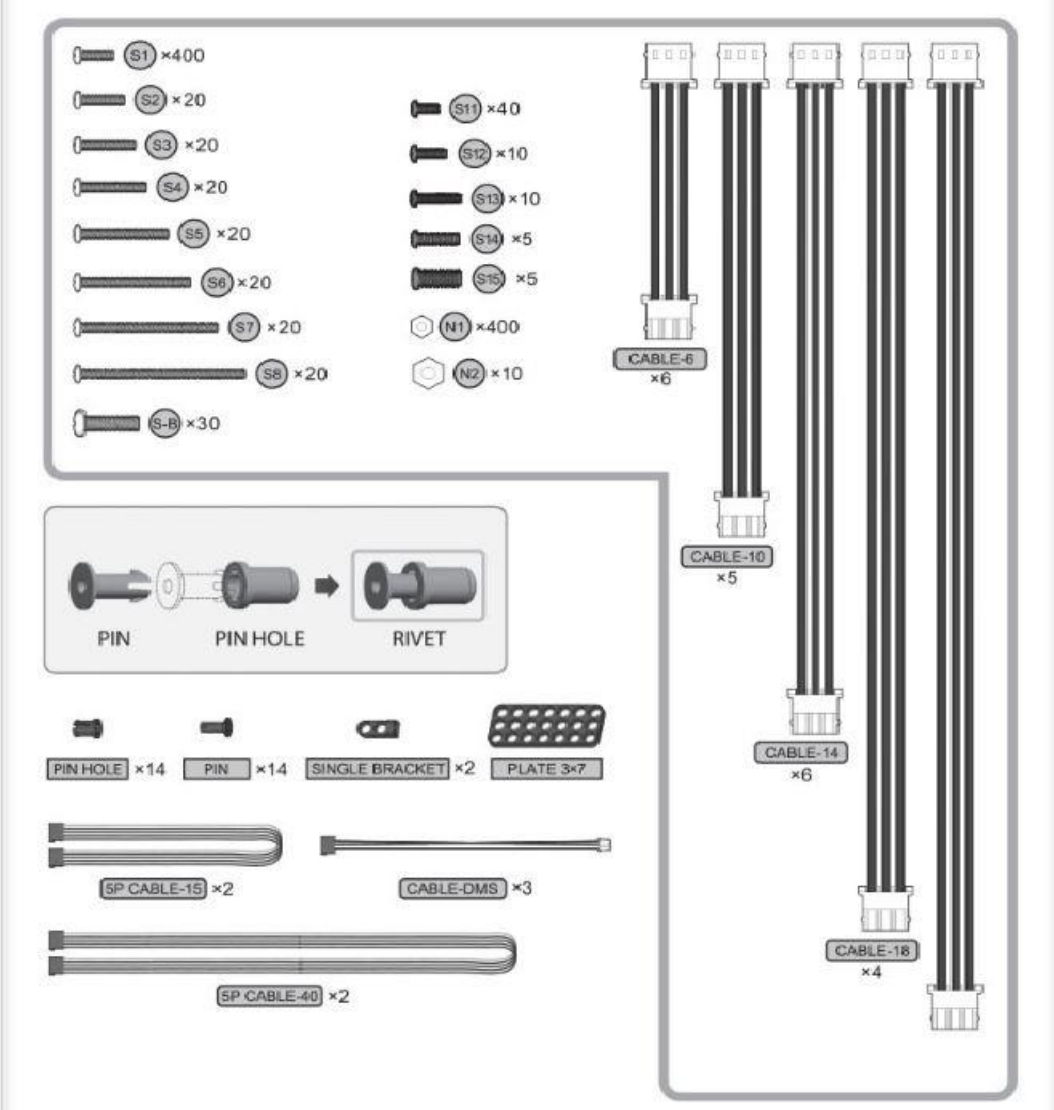


Figura 7-1. Componentes Kit Bioloid Premium

7.2 Anexo II. Conexión de Arbotix-M para la programación:

I. Descargar e Instalar Arduino IDE:

Es necesario descargar el software ARDUINO 1.0.6, ya que las bibliotecas están preparadas para este software.

- Pagina de descarga: https://www.arduino.cc/en/Main/OldSoftwareReleases

II. Instalar los Controladores FTDI.:

Esto permitirá usar el cable FTDI-USB usado para la programación y la comunicación serie.

- Página de descarga: http://www.ftdichip.com/Drivers/VCP.htm
- Guía de instalación de FTDI: http://www.ftdichip.com/Support/Documents/InstallGuides.htm

III. Instalar el hardware Arbotix-M y los archivos de la Biblioteca:

Necesitaremos descargar las biblitoecas. Será un archivo comprimido con 3 carpetas, (hardware, bibliotecas y ArbotiX Sketches). Será necesario copiar las tres en la carpeta de usuario 'Arduino', (no es la ubicación del Arduino IDE). Según su Sistema operative, la carpeta puede estar en:

Windows XP → Mis documentos \ Arduino \

Windows Vista o posteriors → Documentos \ Arduino \

Mac / Linux → ~ / Documentos / Arduino /

IMPORTANTE: Si ya dispone de estas carpetas, copie el contenido de dentro de cada carpeta en su análoga, sin sobreescribir las ya existentes.

- Enlace de descarga de bibliotecas: https://github.com/trossenrobotics/arbotix/archive/master.zip

IV. Conectar el Arbotix Robocontroller a su computadora y a los Actuadores.

La orientadocion del cable FTDI es muy importante, por lo que se debe tener especial precaución al conectarlo. Se deberá conectar el pin correspondiente al cable negro con el pin de la placa con la marca 'BLK'.

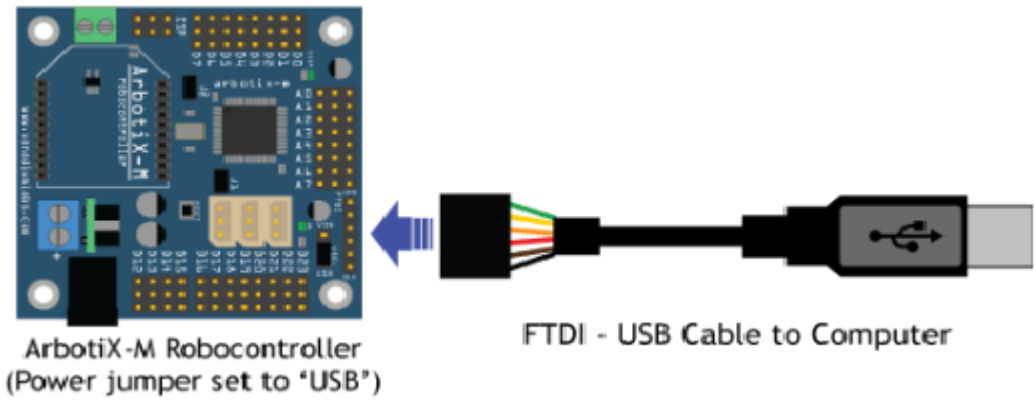


Figura 7-2. Conexión Arbotix con Programador FTDI-USB

Por otro lado, se deberá puentear con el "Jumper" la alimentación, conectando el pin marcado con 'VIN' y el pin central (que se trata de la referencia), y, por consiguiente, dejando libre el marcado con 'USB'. Esto permitirá conectar la Fuente de tensión al controlador.

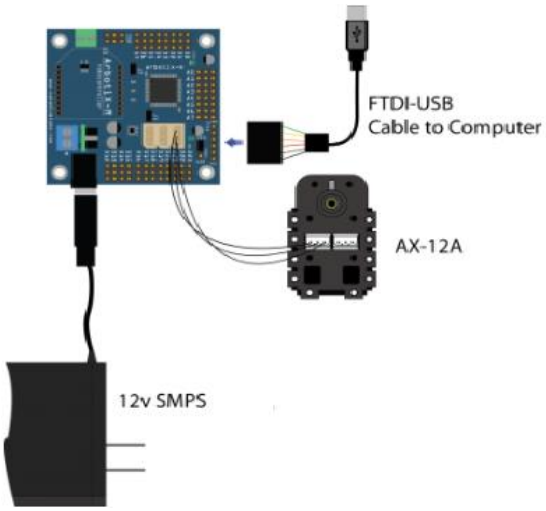


Figura 7-3. Conexión Completa Arbotix - M

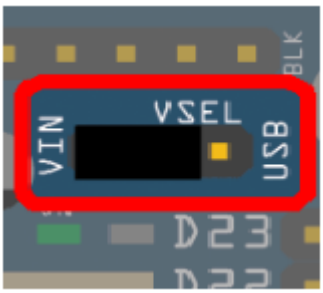


Figura 7-4. Jumper VIN

Por último, se debe conectar el bus TTL del primero motor de la cadena al controlador.

Una vez terminados los preparativos, puede empezar a programar el Arbotix hacienda uso de la IDE Arduino. Esto se verá en detalle en el capítulo X.

Para comenzar con la programación del Arbotix, se deben de seguir los siguientes pasos:[11]

[11] Manual de inicio rápido: http://learn.trossenrobotics.com/arbotix/7-arbotix-quick-start-guide

7.3 Anexo III. Configuración Software Slic3r para Impresión:

A continuación, se desarrollan los pasos seguidos para la impresión de cada una de las piezas, previamente diseñadas en Catia

I. Diseño de Piezas:

Todas las piezas se han diseñado sobre una base ovalada. Esto se debe ha que se han intentado minimizar los problemas de "Wrapping", que se acentúan con las esquinas y los tramos rectos. Esta base ovalada suele tener un grosor de entre 0.6 y 1 mm.

II. Crear G-CODE:

Una vez realizado el diseño de la pieza, hay que usar un Slicer o "rebanador" para crear un código con extensión GCODE. Este software tiene como función generar el código necesario para la impresión de la pieza. En este apartado se tendrá que especificar a la máquina todas las configuraciones que son necesarias para que la estructura cumpla sus cometidos de rigidez y precisión.

Seguidamente, se mostrará la configuración elegida para la impresión de nuestras piezas. Para ello se usó el softwre Repetier-Host GEEEtech.

- **Líneas y Perímetros:**

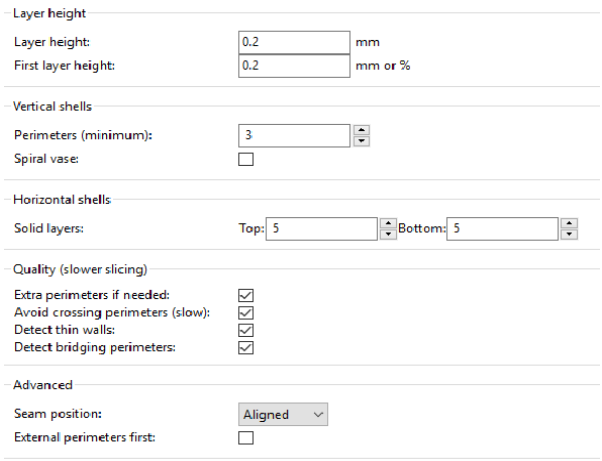


Figura 7-5. Configuración Slicer. Líneas y Perímetros

Layer height: Con este parámetro se controla la altura de capa. Esta es la distancia que el extrusor subirá con respecto a la base cada vez que acabe una capa.
First Layer Height: Este parámetro es igual al anterior con la única diferencia de que afectará solo a la primera capa de impresión.
Perimeters (mínimum): Como su nombre indica, este valor permite modificar el número de perímetros o paredes que tendrá la pieza, de esta forma podremos tener piezas con paredes más gruesas o más finas.
Spiral vase: Algo muy útil si queremos hacer piezas sin relleno, únicamente con el grosor de un perímetro. Aunque para el caso que nos ocupa es contraproducente.
Solid Layers: Hace referencia al número de capas totalmente sólidas que se realizarán en la pieza

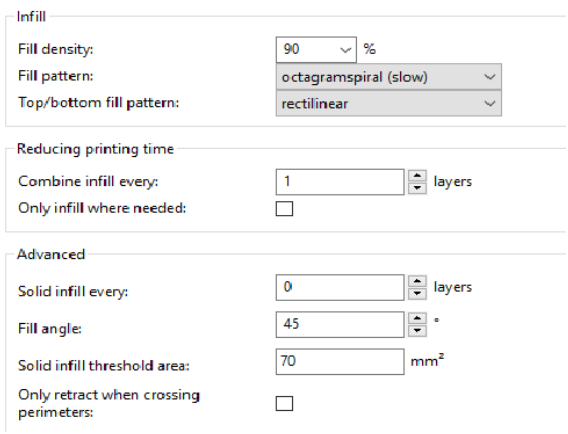


Figura 7-6. Configuración Slicer. Relleno

- Relleno:

Fill density: Este parámetro controla directamente la densidad que tendrá nuestra pieza.
Fill pattern: Aquí le diremos la forma con la que rellenaremos la pieza.
Top/bottom fill pattern: Controla el patrón de relleno de las capas iniciales y finales, este se dejará en rectilíneo para conseguir los mejores acabados.
Reducing printing time: Estos dos parámetros permiten realizar rellenos más rápidos pudiendo alternar entre capas sólidas y huecas, algo que no es recomendable, mantendremos este valor en 1 y desactivaremos la función "*Only infill where needed*" (Solo rellenar cuando sea necesario).
Advanced: Encontraremos parámetros más avanzados

como "*Solid Infill Every*" que nos permite crear una capa totalmente sólida cada cierto número de capas. "*Fill Angle"* es el ángulo con el que se realizará el relleno, dejaremos un valor de 45°. El resto de los parámetros se dejan tal y como vienen predefinidos.

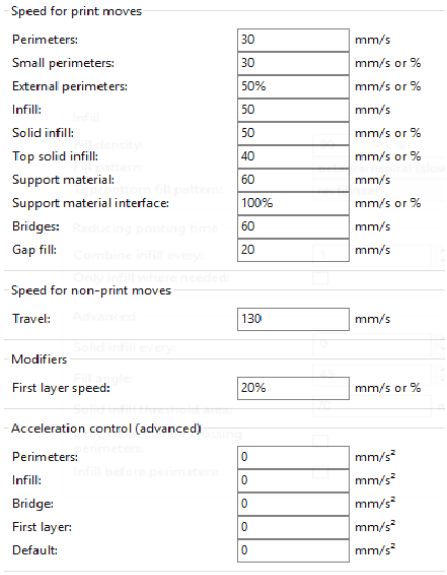
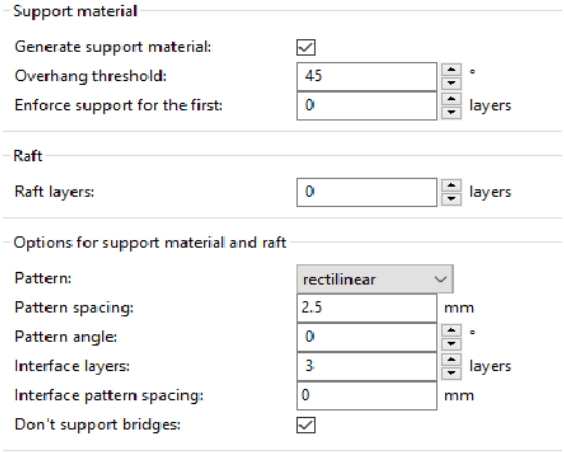


Figura 7-7. Configuración Slicer. Velocidades

-**Velocidades:**

Perimeters: Las tres primeras casillas hacen referencia a la velocidad con la que se imprimirán los perímetros.
Small perimeters: Permite ajustar la velocidad de los perímetros pequeños donde necesitaremos una menor velocidad.
External perimeters: Decide la velocidad del perímetro exterior de nuestra pieza, el que veremos, es conveniente que lo realicemos con una menor velocidad a la general.
Infill: Aquí controlaremos la velocidad de impresión del relleno de la pieza.
Solid infill: Hace referencia a la velocidad de impresión de las capas con un relleno del 100%, excluyendo a la primera y última capa.
Top solid infill: Permite ajustar la velocidad de la última capa de impresión
Support material / Support material interface: Velocidad con la que se imprimirán los soportes de la pieza en el caso de los hayamos activado.
Bridges: Esta es la velocidad con la que se realizarán los puentes, es decir, cuando el extrusor tenga que estirar el filamento de un punto a otro en el aire sin ningún soporte debajo. Al no tener ningún punto de soporte el filamento fundido colgará y se deformará.
Travel: Este parámetro controla la velocidad con la que se mueve el extrusor de un punto a otro cuando no tiene que extruir plástico, por lo tanto, puede subirse.
First layer speed: Permite decidir la velocidad con la que imprimiremos la primera capa, es la que estará en contacto con la superficie de impresión y deberemos hacerla despacio para que la pieza se enganche bien y no tengamos ningún problema con plásticos como ABS que tiende a deformarse.



-**Soportes:**

Generate support material: Esta casilla es la que se activa para que directamente Slic3r genere soportes en nuestras piezas, este proceso lo hará de forma automática a partir de los valores se configuren.
Overhang threshold: Permite marcar en grados la inclinación máxima que la impresora puede imprimir sin realizar soportes.
Enforce support for the first: Este parámetro nos permite forzar la creación de soportes en las primeras capas independientemente de la inclinación que tengan.

Figura 7-8. Configración Slicer. Soportes

Raft layers: El raft es una cama sobre la que se imprimirá la pieza, actúa como soporte, el valor que pongamos configurará el número de capas que generará de este tipo. Normalmente se utiliza en piezas que tengan dificultades para adherirse a la superficie de impresión o bien piezas donde la base no es plana. En este caso, se diseñaron las piezas con este soporte lo suficientemente grande como para luego poder juntar el ABS Juice.
Patter: Aquí se define el patrón con el que se generarán los soportes. Normalmente se utiliza el patrón rectilíneo ya que es más fácil de retirar.
Pattern spacing: Define la separación del patrón de soporte.
Pattern angle: Es el ángulo con el que se imprimirá el soporte.

- **Filamento:**

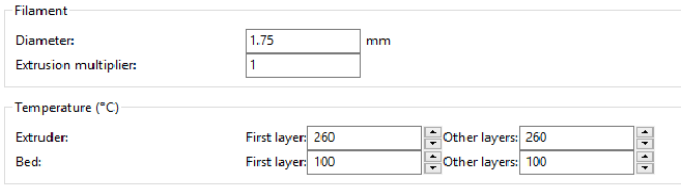


Figura 7-9. Configuración Slicer. Filamento

Diameter: Se usa este parámetro para configurar el diámetro de nuestro filamento.
Extrusion multiplier: Es la relación entre el engranaje pequeño y el engranaje grande del extrusor. Recomendamos no modificar este valor ya que este parámetro lo define Slic3r a través del firmware de la impresora. Dejaremos como valor 1.

Temperature Extruder: Temperatura del extrusor. Se puede poner temperaturas diferentes para la primera y las siguientes.
Temperature Bed: La temperatura de la base caliente también varía en función del material a imprimir.

III. Calibración de Impresora:

Una vez tenemos el archivo GCODE, se debe configurar la impresora para la realización de la pieza. Lo primero que se debe hacer es ejecutar el comando de Auto-Home. Tras esto, hay que calibrar la cama con el extrusor. Para ello, se irá deslizando hasta ponerlo en las cuatro esquinas de la cama y el centro. Se deberá pasar un pedazo de papel entre el extrusor y la cama, notándose una resistencia (no debe de entrar de manera holgada). Cuando se note en las cuatro esquinas y el centro aproximadamente la misma resistencia al pasar el papel, estará bien equlibrada y a la altura correcta. Una vez calibrada, se le echa laca de fijación en abundancia en el cristal térmico.

IV. Impresión:

Se vuelve a hacer el Auto-Home, se introduce la tarjeta SD con el archivo GCODE y se ejecuta la intrucción para comenzar a imprimir. Primero se calentará la cama y una vez llegue a la temperatura adecuada, lo hará el extrusor. Cuando todas las temperaturas sean las adecuadas comenzará automáticamente la impresión.

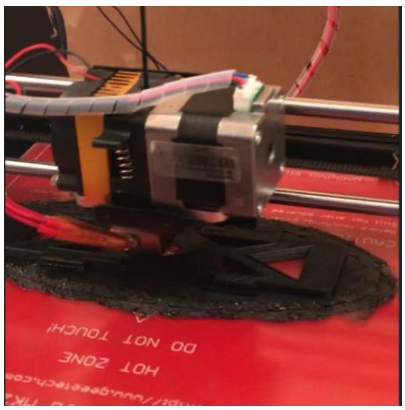


Figura 7-10. Aplicación ABS Juice

V. Aplicación de ABS Juice:

Este paso es de vital importancia para evitar el "Wraping[12]". Una vez la pieza haya imprimido el primer milímetro de la pieza, habrá terminado de hacer la base. Será el momento de pausar la impresión. Cuando haya parado, se aplicará el ABS Juice, una mezcla de ABS (40%) y acetona (60%). Esto creará una sustancia pegajosa que se extenderá en el contorno de la base de la pieza, haciendo que se pegue al cristal. Una vez echada, reanudar la impresión.

VI. Despegue de Pieza y Acabado Final:

Cuando la impresión termine, dejar enfriar la pieza. Cuando lo haya hecho, con un cuter despegar la pieza del cristal y recortar el sobrante de la base. Se lijan los contornos para conseguir un buen acabado.

[12] Wrapping: El ABS se imprime a unos 260 ºC. Al enfriarse rapidamenteal ser extruido, este se deforma y se contrao un poco. Cuando se realizan objetos con grandes superficies, los filamentos intermedios tienden a tirar de las esquinas, provocando que la pieza se despegue del cristal templado. Esto hace que la pieza se deforme y pierda su funcionalidad. Es uno de los principales problemas y requirió de bastante tiempo el poder solucionarlo.

7.4 Anexo IV. Manual de Preparación de RaspBerry Pi.

Para poder implementar el procedimiento de visión artificial, primero hay que configurar el software de la RaspBerry. Para ello habrá que instalar el sistema operativo. Una vez realizado, se deberá realizar una conexión remota por SSH. Para concluir la configuración, se debe instalar las librerías Open CV que permite realizar el procesamiento de imagen.

7.4.1 Instalación de Sistema Operativo Raspbian Stretch.

I. Instalación de Software:

Para la configuración de la Raspberry necesitaremos los siguientes programas software instalados en el PC:

- Un programa para formatear tarjetas SD (*SD Card Formater*).
- Software para montar imágenes en tarjetas SD (*Etcher*).
- Cliente SSH para conectar a servidores remotos y ejecutar comandos (*Putty*).
- Cliente VCN para conectar a servidores remotos y tener visión de la pantalla (*VCN Viewer*).
- Software para escanear direcciones IP (*Wirelles Netwoek Watcher*).

II. Formateo de tarjeta SD:

a. Insertar la tarjeta microSD en el ordenador.
b. Hacer clic en el icono del programa SD Card Formate.
c. Saldrá una pantalla como la que tenemos en la imagen. Se debe seleccionar el puerto en el cual tenemos insertado la tarjeta y clicamos en Format.
d. Aparecerá un cuadro de diálogo en el cual pregunta si estamos seguros de que se desea realizar el formateo de la tarjeta. Se deberá clicar en Si.
e. Comenzará el formateo de la tarjeta. Una vez terminado, aparecerá una nueva ventana indicando el resumen del proceso. Para finalizar, pulsar Aceptar.

III. Descargar y montar Sistema operativo:

a. Descargar el sistema operativo RASPBIAN STRETCH CON ESCRITORIO. Para ello solo tenemos que ir al siguiente enlace y descargarlo, haciendo clic en Descargar ZIP.
b.
ENLACE: https://www.raspberrypi.org/downloads/raspbian/

c. Hacer clic en el icono de Etcher. Aparecerá una ventana como la que se observa en la Imagen.
d. Seleccionar el archivo raspbian-stretch y pulsar en Abrir.
e. Para comenzar el montaje de la imagen, hacer clic en Flash!
f. Una vez concluido, pulsar en Flash Another para terminar.

IV. Crear un archivo sin extensión llamado SSH:

Tras la instalación del sistema operativo, habrá que ir a la root donde se ha instalado el sistema operativo en la tarjeta SD. Una vez dentro, se deberá crear un archivo sin extensión y que no contenga ninguna información, que se llamará SSH.

Para ello se debe abrir el bloc de notas y guardar un archivo en blanco en el directorio anteriormente indicado. Una vez hecho esto, habrá que dirigirse al archivo creado, hacer clic con el botón derecho y renombrar el archivo, borrando la extensión del final del nombre y pulsar aceptar. Aparecerá una ventana que dirá si estamos seguro de querer quitar la extensión al archivo, la cual se debe volver a aceptar.

Una vez realizado esto, extraer con seguridad la tarjeta SD del ordenador.

7.4.2 Conexión Remota.

Una vez terminada la instalación del sistema operativo, debemos insertar la tarjeta microSD en la Raspberry, junto con el cable RJ45, que irá del router a la placa y la fuente de alimentación de 5 V. Finalizada las conexiones, se deberá esperar unos minutos para que arranque el sistema operativo. Tras esto, proceder de la siguiente manera:

a. Abrir el programa *Wirelles Netwoek Watcher* para detectar la IP de la Raspberry. Para ello debemos hacer clic en escanear (botón play de color verde).
b. Una vez detectada la IP de la Raspberry, podemos abrir Putty y poner la dirección IP correcta para poder acceder a la misma a través de SSH.
c. Una vez puesta la dirección, le damos a Open. Se abrirá un terminal de texto. Lo primero que se debe de hacer es registrarnos. Nos pedirá el nombre de usuario, el cuál viene por defecto con el nombre de "pi". Tras esto se deberá escribir la contraseña, que es "raspberry", la cual no se verá cuando se escriba por temas de seguridad.
d. Tras esto debemos escribir por el terminal: *sudo raspi-config*
e. Aparecerá una pantalla como la que se observa en la siguiente imagen. Se deberá ir a la opción 5 y pulsar Enter.
f. Tras esto hay que ir a la opción de SSH y pulsar de nuevo Enter.
g. Aparecerá una pantalla que pregunta si se esta seguro de realizar dicha acción. Habrá que pulsar en Yes. Una vez hecho esto, se tendrá activado el SSH.

h. Al aceptar volverá a aparecer la imagen X del paso e. Se deberá volver a realizar los pasos para hacer lo mismo con la conexión VCN que nos permitirá tener un escritorio virtual.

7.4.3 OpenCV y Librerías.[13]

I. Expandir sistema de archivos.

Para ello hay que escribir en el terminal:

$ sudo raspi-config

Saldrá el menú principal y se deberá seleccionar "Opciones Avanzadas". Seguidamente, seleccionar "Expandir sistemas de archivos".

Una vez terminado, reiniciar el sistema exribiendo en el terminal:

$ sudo reboot

II. Instalar Dependencias:

a) Actualizar paquetes existentes.

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

b) Instalar herramienta CMake:

$ sudo apt-get install build-essential cmake pkg-config

c) Instalar paquetes de E/S de imágenes y vídeos:

$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev

$ sudo apt-get install libxvidcore-dev libx264-dev

d) Instalar Biblioteca de desarrollo GTK:

[13] Tutorua de Instalación: https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/

$ sudo apt-get install libgtk2.0-dev libgtk-3-dev

e) Instalar dependencias que optimizarán OpenCV:

$ sudo apt-get install libatlas-base-dev gfortran

f) Instalar archivos de cabecera de Python 3:

$ sudo apt-get install python2.7-dev python3-dev

III. Descargar el Código Fuente de OpenCV:

g) Descargar OpenCV desde su repositorio:

$ cd ~

$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3.3.0.zip

$ unzip opencv.zip

$ wget -O opencv_contrib.zip https://github.com/Itseez/opencv_contrib/archive/3.3.0.zip

$ unzip opencv_contrib.zip

IV. Compilar OpenCV en Python:

h) Instalar administrador de paquetes pip:

$ wget https://bootstrap.pypa.io/get-pip.py

$ sudo python get-pip.py

$ sudo python3 get-pip.py

i) Instalar Entorno Virtual:

$ sudo pip install virtualenv virtualenvwrapper

$ sudo rm -rf ~/.cache/pip

j) Actualización de perfil:

$ source ~/.profile

k) Crear entorno virtual en python:

$ mkvirtualenv cv -p python3

V. Instalar NumPy:

l) Instalar NumPy que ayuda con el procesamiento numérico:

$ pip install numpy

VI. Compilar e Instalar OpenCV:

m) Verificar Entorno Virtual:

$ workon cv

n) Configurar la compilación:

$ cd ~/opencv-3.3.0/

$ mkdir build

$ cd build

```
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
  -D CMAKE_INSTALL_PREFIX=/usr/local \
  -D INSTALL_PYTHON_EXAMPLES=ON \
  -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.3.0/modules \
  -D BUILD_EXAMPLES=ON ..
```

o) Configurar el tamaño de espacio de intercambio. Para ello abrir archivo / etc / dphys - swapfile y luego edite la variable CONF_SWAPSIZE

 # set size to absolute value, leaving empty (default) then uses computed value
 # you most likely don't want this, unless you have an special disk situation
 # CONF_SWAPSIZE=100
CONF_SWAPSIZE=1024

p) Reiniciar el servicio de intercambio:

$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start

q) Compilar OpenCV: Este paso puede durar un par de horas.

$ make -j4

r) Instalar OpenCV:

$ sudo make install
$ sudo ldconfig

VII. Concluir Instalación:

s) Verificación de Instalación:

$ ls -l /usr/local/lib/python3.5/site-packages/
total 1852
-rw-r--r-- 1 root staff 1895932 Mar 20 21:51 cv2.cpython-34m.so

t) Cambiar nombre a paquete:

$ cd /usr/local/lib/python3.5/site-packages/
$ sudo mv cv2.cpython-35m-arm-linux-gnueabihf.so cv2.so
$ cd ~/.virtualenvs/cv/lib/python3.5/site-packages/
$ ln -s /usr/local/lib/python3.5/site-packages/cv2.so cv2.so

u) Verificación de Instalación:

$ ls -l /usr/local/lib/python3.5/site-packages/
total 1852
-rw-r--r-- 1 root staff 1895932 Mar 20 21:51 cv2.cpython-34m.so

7.5 Anexo V: Código Control Brazo Robot Industrial Arduino:

```
#include <MatrixMath.h>
#include <ax12.h>
#include <stdio.h>
#include <math.h>
#include <BioloidController.h>

BioloidController bioloid = BioloidController(1000000); // Velocidad del controlador.

//-------------------------------------------------------
 VARIABLES DE CARACTER GENERAL :-------------------------------------------------------
--------//
int i = 0;
int j = 0;
int AX_GOAL_POSITION = 30;
int Continuar = 0;
int contador = 0;
int FLAG = 0;
int punto=0;
int cont_total=0;
int posicion=0;
int orientacion=0;
double iteracion=0;

//-------------------------------------------------------
 VARIABLES DE LECTURA POR PUERTO SERIE:-----------------------------------------------
---------------//

char InCadena;
String cadena = "";
String cadenaleida = "";
int esperaleer = 1;

//-------------------------------------------------------- VARIABLES DE CONTROL :--------
--------------------------------------------------------//

float sat=2;

float kpx = 8;
float kpy = 8;
float kpz = 15;
float kpa = 20;
float kpb = 20;
float kpg = 20;

float kix = 0.30;
float kiy = 0.30;
float kiz = 7.00;
float kia = 0.10;
float kib = 0.10;
float kig = 0.10;

float t = 0.03;

float ex1=0;
float det=0;
float eps=0.001;
float k0=0.001;
float k=0;

//-------------------------------------------------------
  VARIABLES PARA CALCULO ERROR INTEGRAL :-----------------------------------------------
-----------------//

float Sxant = 0;
float Sx = 0;
float Syant = 0;
```

```
float Sy = 0;
float Szant = 0;
float Sz = 0;
float Saant = 0;
float Sa = 0;
float Sbant = 0;
float Sb = 0;
float Sgant = 0;
float Sg = 0;

//------------------------------------------------------ ESCRITURA EN MOTORES :--------
-------------------------------------------------------//

// Variables para escritura sincrona

int vectpos[8][2] = { { 2, 0 },{ 4, 0 },{ 6, 0 },{ 8, 0 },{ 10, 0 },{ 12, 0 },{ 14, 0
},{ 18, 0 } };

int vectvel[8][2] = { { 2, 50 },{ 4, 50 },{ 6, 50 },{ 8, 50 },{ 10, 50 },{ 12, 50 },{
14, 50 },{ 18, 50 } };

// Variables para escritura de dos motores

int qdmV4 = 0;
int qdmV6 = 0;
int arti2[2][2] = { { 4, 0 },{ 6, 0 } };

int qdmV8 = 0;
int qdmV10 = 0;
int arti3[2][2] = { { 8, 0 },{ 10, 0 } };

//------------------------------------------------------ POSICIONES CARTESIANAS :------
-------------------------------------------------------//

// Posicion Deseada:
int xleida = 0;
int yleida = 0;
int zleida = 0;
int aleida = 0;
int bleida = 0;
int gleida = 0;

float Xdes = 0;
float Ydes = 0;
float Zdes = 0;
float Ades = 0;
float Bdes = 0;
float Gdes = 0;

// Posicion Actual:
float xact = 0;
float yact = 0;
float zact = 0;
float aact = 0;
float bact = 0;
float gact = 0;

//------------------------------------------------------ ERRORES:----------------------
----------------------------------------//

// Error de Posicion Actual:
float error[6][1] = { { 0 },{ 0 },{ 0 },{ 0 },{ 0 },{ 0 } }; // Error de posciion de
destino y posicion actual
float errorkp[6][1] = { { 0 },{ 0 },{ 0 },{ 0 },{ 0 },{ 0 } }; // Error por KP
float errorki[6][1] = { { 0 },{ 0 },{ 0 },{ 0 },{ 0 },{ 0 } }; // Error por KI

float errorkpi[6][1] = { { 0 },{ 0 },{ 0 },{ 0 },{ 0 },{ 0 } }; // Error  KP + Error
KI
```

```
float errorabs[6][1] = { { 1 },{ 1 },{ 1 },{ 1 },{ 1 },{ 1 } }; // Error en valor
absoluto

float errorposicion = 5; // Media del error de cada una de las posiciones.
float errororientacion=5;  // Media del error de cada una de las orientaciones.

//-----------------------------------------------------
 POSICIONES EN COORDENADAS ARTICULARES :-----------------------------------------
----------------//

// Posicion actual de los motores 0 a 1023.
int q1m = 0;
int q2m = 0;
int q22m = 0;
int q3m = 0;
int q33m = 0;
int q4m = 0;
int q5m = 0;
int q6m = 0;
int qpm = 0;

// Posicion actual de los motores en radianes.
float q1 = 0;
float q2 = 0;
float q3 = 0;
float q4 = 0;
float q5 = 0;
float q6 = 0;
float qp = 0;

// Posicion deseada de los motores 0 a 1023.
int q1mdes = 0;
int q2mdes = 0;
int q3mdes = 0;
int q4mdes = 0;
int q5mdes = 0;
int q6mdes = 0;
int qpmdes = 0;

// Posicion deseada de los motores en radianes.
float q1des = 0;
float q2des = 0;
float q3des = 0;
float q4des = 0;
float q5des = 0;
float q6des = 0;
float qpdes = 0;

// incremento de Q
float incq1 = 0;
float incq2 = 0;
float incq3 = 0;
float incq4 = 0;
float incq5 = 0;
float incq6 = 0;


//------------------------------------------------------ VELOCIDADES:------------------
-------------------------------------------//

float qd[6][1] = { { 0 },{ 0 },{ 0 },{ 0 },{ 0 },{ 0 } }; // Se guardan las
velocidades resultantes de Jc*errrorKP
float qds[6][1] = { { 0 },{ 0 },{ 0 },{ 0 },{ 0 },{ 0 } }; // Signo de la velocidad;
float qdf[6][1] = { { 0 },{ 0 },{ 0 },{ 0 },{ 0 },{ 0 } }; // Se guardan las
velocidades finales.
float qdr[6][1] = { { 0 },{ 0 },{ 0 },{ 0 },{ 0 },{ 0 } }; // Velocidades anteriores
en RPM
int qdm[6][1] = { { 0 },{ 0 },{ 0 },{ 0 },{ 0 },{ 0 } }; // Velocidades en MOTOR (0-
1023 || 1024-2047);
```

```
int velmax[6][1] = { { 300 },{ 800 },{ 800 },{ 500 },{ 500 },{ 500 } }; // Velocidad
de saturacion de los motores.

//-------------------------------------------------------
  VARIABLES EN FORMA DE MATRIZ PARA JACOBIANO :-------------------------------------
-----------------------//

// Matriz Jacobiana.
float J[6][6] = {
         { 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 } };

// Jacobiana traspuesta.
float Jt[6][6] = {
         { 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 } };

// Jacobiana por Jacobiana traspuesta
float JJ[6][6] = {
         { 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 } };

// Matriz identidad con parametro K
float IK[6][6] = {
         { 0.01,0,0,0,0,0 },
{ 0,0.01,0,0,0,0 },
{ 0,0,0.01,0,0,0 },
{ 0,0,0,0.01,0,0 },
{ 0,0,0,0,0.01,0 },
{ 0,0,0,0,0,0.01 } };

// Matriz JJ multiplicada por matriz IK
float JJJ[6][6] = {
         { 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 } };

// Inversa de la matris JJ
float JJinv[6][6] = {
         { 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 } };

// Jacobiano Cruz Final.
float Jc[6][6] = {
         { 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
```

```
{ 0,0,0,0,0,0 } };

float Jcn[6][6] = {
        { 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 },
{ 0,0,0,0,0,0 } };

// the setup function runs once when you press reset or power the board
void setup() {
        pinMode(0, OUTPUT);

        //INICIALIZACION DE VARIABLES:
        cadena.reserve(200);
        cadenaleida.reserve(200);
        esperaleer = 0;

        // HABILITANDO COMUNICACION SERIE: //
        Serial.begin(9600);
        delay(1000);
        Serial.println("############################");
        Serial.println("COMUNICACION SERIE HABILITADA.");
        Serial.println("############################");

        Serial.println("############################");
        Serial.println("Activamos el torque de todos los motores");
        Serial.println("############################");

        dxlTorqueOnAll(); // Funcion para activar todos los torques

        delay(1000);

}

// the loop function runs over and over again until power down or reset
void loop() {

        FLAG = 0;
       posicion=0;
       orientacion=0;
       iteracion=0;

        LeePosiciones();
        CalculaInversa();
        EligePosicion();
        CalculaInversa();
        CalculaError();


        do
        {
                CalculaJacobiano();
               CalculaIK();
                CalculaJC();
                CalculaVelDestino();
                CalculaPosDestino();
                CalculaActuacionMotores();
                Actua();

                LeePosiciones();
                CalculaInversa();
                CalculaError();

               if (iteracion<300) {iteracion++;}
               else
               {
                 FLAG=1;
```

```
                Serial.println("OverFLow");
              }

        } while (FLAG == 0);

        if (punto<4)
          {
            punto++;
            cont_total++;
            delay(500);
          }
        else if (cont_total<40){punto=1;}
        else
        {
          punto=0;
          cont_total=0;
        }


}


// FUNCION PARA LEER POSICIONES DE LOS MOTORES.
void LeePosiciones() {

        q1m = dxlGetPosition(2);
        if (q1m<1)
        {
                q1m = dxlGetPosition(2);
        }
        q1 = (q1m - 512)*0.0051;


        q2m = dxlGetPosition(4);
        if (q2m<1)
        {
                q2m = dxlGetPosition(4);
        }

        q22m = dxlGetPosition(6);
        if (q22m<1)
        {
                q22m = dxlGetPosition(6);
        }

        q2 = (q2m - 512)*0.0051;

        q3m = dxlGetPosition(8);
        if (q3m<1)
        {
                q3m = dxlGetPosition(8);
        }

        q33m = dxlGetPosition(10);
        if (q33m<1)
        {
                q33m = dxlGetPosition(10);
        }
        q3 = (q3m - 512)*0.0051;

        q4m = dxlGetPosition(12);
        if (q4m<1)
        {
                q4m = dxlGetPosition(12);
        }
        q4 = (q4m - 512)*0.0051;

        q5m = dxlGetPosition(14);
        if (q5m<1)
```

```
        {
                q5m = dxlGetPosition(14);
        }
        q5 = -(q5m - 512)*0.0051;

        q6m = dxlGetPosition(18);
        if (q6m<1)
        {
                q6m = dxlGetPosition(18);
        }
        q6 = (q6m - 512)*0.0051;
}



// CALCULO DE  X, Y, Z, A, B y G ACTUAL E IMPRIME  -
void CalculaInversa() {

        xact = 0.085*sin(q1 + q2) - 0.085*sin(q1 - 1.0*q2) + 0.066*cos(q1 -
1.0*q2)*sin(q3) - 0.066*sin(q1 - 1.0*q2)*cos(q3) + 0.075*sin(q1 + q2 + q3)*cos(q5) +
0.075*sin(q2 - 1.0*q1 + q3)*cos(q5) + 0.066*cos(q1 + q2)*sin(q3) + 0.066*sin(q1 +
q2)*cos(q3) - 0.15*sin(q1)*sin(q4)*sin(q5) + 0.075*cos(q1 + q2 + q3)*cos(q4)*sin(q5) +
0.075*cos(q2 - 1.0*q1 + q3)*cos(q4)*sin(q5);

        yact = 0.085*cos(q1 - 1.0*q2) - 0.085*cos(q1 + q2) + 0.066*sin(q1 -
1.0*q2)*sin(q3) - 0.075*cos(q1 + q2 + q3)*cos(q5) + 0.075*cos(q2 - 1.0*q1 +
q3)*cos(q5) - 0.066*cos(q1 + q2)*cos(q3) + 0.066*cos(q1 - 1.0*q2)*cos(q3) +
0.066*sin(q1 + q2)*sin(q3) + 0.075*sin(q1 + q2 + q3)*cos(q4)*sin(q5) - 0.075*sin(q2 -
1.0*q1 + q3)*cos(q4)*sin(q5) + 0.15*cos(q1)*sin(q4)*sin(q5);

        zact = 0.13*cos(q2 + q3) + 0.17*cos(q2) + cos(q5)*(0.15*cos(q2 + q3)) -
1.0*sin(q5)*(+0.15*sin(q2 + q3)*cos(q4)) + 0.096;

        aact = atan((sin(q4)*sin(q5)) / (cos(q4)*sin(q5)));

        bact = atan(sqrt((sin(q4)*sin(q5))*(sin(q4)*sin(q5)) + (cos(q4)*sin(q5)) *
(cos(q4)*sin(q5))) / (cos(q5)));

        gact = atan((1.0*sin(q5)*sin(q6)) / (1.0*cos(q6)*sin(q5)));

}

// ELECCION DE POSICIONES DE DESTINO (PUESTO SERIE)
void EligePosicion() {


       if (punto==0)
       {
         Xdes= 0.1702;
         Ydes= 0.0183;
         Zdes= 0.3948;
         Ades= -0.0612;
         Bdes= 0.7293;
         Gdes= -1.4841;

         Serial.println("############################");
         Serial.println("- Introduce algo para continuar: ");
         Serial.println("----------------------------");

         Leer();
         Continuar = cadenaleida.toInt();
         cadenaleida = "";
         esperaleer = 0;
         Serial.print("Has pulsado continuar:  ");
         Serial.println(Continuar);
         Serial.println("");
       }

       if (punto==1)
```

```
        {
          Xdes= 0.1702;
          Ydes= 0.0183;
          Zdes= 0.3948;
          Ades= -0.0612;
          Bdes= 0.7293;
          Gdes= -1.4841;
        }

        else if (punto==2)
        {
          Xdes= 0.2729;
          Ydes= 0.0787;
          Zdes= 0.2367;
          Ades= 0.4998;
          Bdes= 0.6987;
          Gdes= -0.0306;
        }

        else if (punto==3)
        {
          Xdes= 0.2505;
          Ydes= 0.0744;
          Zdes= 0.0948;
          Ades= -0.0612;
          Bdes= 0.7293;
          Gdes= -1.4841;
        }



}

// CALCULO DE ERRORES.
void CalculaError() {


        error[0][0] = (Xdes - xact)/(1+(20*((Xdes - xact)*(Xdes - xact))));
        error[1][0] = (Ydes - yact)/(1+(20*((Ydes - yact)*(Ydes - yact))));
        error[2][0] = (Zdes - zact)/(1+(100*((Zdes - zact)*(Zdes - zact))));
        error[3][0] = (Ades - aact)/(1+(10*((Ades - aact)*(Ades - aact))));
        error[4][0] = (Bdes - bact)/(1+(10*((Bdes - bact)*(Bdes - bact))));
        error[5][0] = (Gdes - gact)/(1+(10*((Gdes - gact)*(Gdes - gact))));

        errorabs[0][0] = abs(Xdes - xact);
        errorabs[1][0] = abs(Ydes - yact);
        errorabs[2][0] = abs(Zdes - zact);
        errorabs[3][0] = abs(Ades - aact);
        errorabs[4][0] = abs(Bdes - bact);
        errorabs[5][0] = abs(Gdes - gact);

        errorkp[0][0] = error[0][0] * kpx;
        errorkp[1][0] = error[1][0] * kpy;
        errorkp[2][0] = error[2][0] * kpz;
        errorkp[3][0] = error[3][0] * kpa;
        errorkp[4][0] = error[4][0] * kpb;
        errorkp[5][0] = error[5][0] * kpg;

        Sxant = Sx;
        Sx = Sxant + (error[0][0] * t);
        if (Sx > sat)  { Sx = sat; }
        if (Sx < -sat) { Sx = -sat; }

        Syant = Sy;
        Sy = Syant + (error[1][0] * t);
        if (Sy > sat)  { Sy = sat; }
        if (Sy < -sat) { Sy = -sat; }

        Szant = Sz;
```

```
        Sz = Szant + (error[2][0] * t);
        if (Sz > sat)  { Sz = sat; }
        if (Sz < -sat) { Sz = -sat; }

        Saant = Sa;
        Sa = Saant + (error[3][0] * t);
        if (Sa > sat)  { Sa = sat; }
        if (Sa < -sat) { Sa = -sat; }

        Sbant = Sb;
        Sb = Sbant + (error[4][0] * t);
        if (Sb > sat)  { Sb = sat; }
        if (Sb < -sat) { Sb = -sat; }

        Sgant = Sg;
        Sg = Sgant + (error[5][0] * t);
        if (Sg > sat)  { Sg = sat; }
        if (Sg < -sat) { Sg = -sat; }

        errorki[0][0] = Sx * kix;
        errorki[1][0] = Sy * kiy;
        errorki[2][0] = Sz * kiz;
        errorki[3][0] = Sa * kia;
        errorki[4][0] = Sb * kib;
        errorki[5][0] = Sg * kig;

       errorkpi[0][0]=errorkp[0][0]+errorki[5][0];
       errorkpi[1][0]=errorkp[1][0]+errorki[5][0];
       errorkpi[2][0]=errorkp[2][0]+errorki[5][0];
       errorkpi[3][0]=errorkp[3][0]+errorki[5][0];
       errorkpi[4][0]=errorkp[4][0]+errorki[5][0];
       errorkpi[5][0]=errorkp[5][0]+errorki[5][0];

        errorposicion = ((errorabs[0][0] + errorabs[1][0] + errorabs[2][0]) / 3);
       errororientacion = ((errorabs[3][0] + errorabs[4][0] + errorabs[5][0]) / 3);


        if (errorposicion<0.01) { posicion = 1; }
        else { posicion = 0; }

       if (errororientacion<0.1) { orientacion = 1; }
        else { orientacion = 0; }

       if ((posicion==1) && (orientacion==1))
       {
           FLAG=1;
           Serial.println("Goal achieved");
       }
       else { FLAG=0;}
}


// CALCULA JACOBIANO .
void CalculaJacobiano() {


        J[0][0] = 0.15*cos(q4)*sin(q1)*sin(q2)*sin(q3)*sin(q5) -
0.13*cos(q2)*sin(q1)*sin(q3) - 0.13*cos(q3)*sin(q1)*sin(q2) -
0.15*cos(q1)*sin(q4)*sin(q5) - 0.15*cos(q2)*cos(q5)*sin(q1)*sin(q3) -
0.15*cos(q3)*cos(q5)*sin(q1)*sin(q2) - 0.15*cos(q2)*cos(q3)*cos(q4)*sin(q1)*sin(q5) -
0.17*sin(q1)*sin(q2);
        J[0][1] = -0.002*cos(q1)*(66.0*sin(q2)*sin(q3) - 66.0*cos(q2)*cos(q3) -
85.0*cos(q2) - 75.0*cos(q2)*cos(q3)*cos(q5) + 75.0*cos(q5)*sin(q2)*sin(q3) +
75.0*cos(q2)*cos(q4)*sin(q3)*sin(q5) + 75.0*cos(q3)*cos(q4)*sin(q2)*sin(q5));
        J[0][2] = -0.006*cos(q1)*(22.0*sin(q2)*sin(q3) - 22.0*cos(q2)*cos(q3) -
25.0*cos(q2)*cos(q3)*cos(q5) + 25.0*cos(q5)*sin(q2)*sin(q3) +
25.0*cos(q2)*cos(q4)*sin(q3)*sin(q5) + 25.0*cos(q3)*cos(q4)*sin(q2)*sin(q5));
        J[0][3] = -0.15*sin(q5)*(cos(q4)*sin(q1) + cos(q1)*cos(q2)*cos(q3)*sin(q4) -
1.0*cos(q1)*sin(q2)*sin(q3)*sin(q4));
```

```
        J[0][4] = 0.15*cos(q1)*cos(q2)*cos(q3)*cos(q4)*cos(q5) -
0.15*cos(q1)*cos(q2)*sin(q3)*sin(q5) - 0.15*cos(q1)*cos(q3)*sin(q2)*sin(q5) -
0.15*cos(q5)*sin(q1)*sin(q4) - 0.15*cos(q1)*cos(q4)*cos(q5)*sin(q2)*sin(q3);
        J[0][5] = 0;

        J[1][0] = 0.17*cos(q1)*sin(q2) - 0.15*sin(q1)*sin(q4)*sin(q5) +
0.13*cos(q1)*cos(q2)*sin(q3) + 0.13*cos(q1)*cos(q3)*sin(q2) +
0.15*cos(q1)*cos(q2)*cos(q5)*sin(q3) + 0.15*cos(q1)*cos(q3)*cos(q5)*sin(q2) +
0.15*cos(q1)*cos(q2)*cos(q3)*cos(q4)*sin(q5) -
0.15*cos(q1)*cos(q4)*sin(q2)*sin(q3)*sin(q5);
        J[1][1] = -0.002*sin(q1)*(66.0*sin(q2)*sin(q3) - 66.0*cos(q2)*cos(q3) -
85.0*cos(q2) - 75.0*cos(q2)*cos(q3)*cos(q5) + 75.0*cos(q5)*sin(q2)*sin(q3) +
75.0*cos(q2)*cos(q4)*sin(q3)*sin(q5) + 75.0*cos(q3)*cos(q4)*sin(q2)*sin(q5));
        J[1][2] = -0.006*sin(q1)*(22.0*sin(q2)*sin(q3) - 22.0*cos(q2)*cos(q3) -
25.0*cos(q2)*cos(q3)*cos(q5) + 25.0*cos(q5)*sin(q2)*sin(q3) +
25.0*cos(q2)*cos(q4)*sin(q3)*sin(q5) + 25.0*cos(q3)*cos(q4)*sin(q2)*sin(q5));
        J[1][3] = 0.15*sin(q5)*(cos(q1)*cos(q4) - 1.0*cos(q2)*cos(q3)*sin(q1)*sin(q4)
+ sin(q1)*sin(q2)*sin(q3)*sin(q4));
        J[1][4] = 0.15*cos(q1)*cos(q5)*sin(q4) - 0.15*cos(q2)*sin(q1)*sin(q3)*sin(q5)
- 0.15*cos(q3)*sin(q1)*sin(q2)*sin(q5) + 0.15*cos(q2)*cos(q3)*cos(q4)*cos(q5)*sin(q1)
- 0.15*cos(q4)*cos(q5)*sin(q1)*sin(q2)*sin(q3);
        J[1][5] = 0;

        J[2][0] = 0;
        J[2][1] = -0.13*sin(q2 + q3) - 0.17*sin(q2) - 0.15*sin(q2 + q3)*cos(q5) -
0.15*cos(q2 + q3)*cos(q4)*sin(q5);
        J[2][2] = -0.13*sin(q2 + q3) - 0.15*sin(q2 + q3)*cos(q5) - 0.15*cos(q2 +
q3)*cos(q4)*sin(q5);
        J[2][3] = 0.15*sin(q2 + q3)*sin(q4)*sin(q5);
        J[2][4] = -0.15*cos(q2 + q3)*sin(q5) - 0.15*sin(q2 + q3)*cos(q4)*cos(q5);
        J[2][5] = 0;

        J[3][0] = 0;
        J[3][1] = 0;
        J[3][2] = 0;
        J[3][3] = 1;
        J[3][4] = 0;
        J[3][5] = 0;

        J[4][0] = 0;
        J[4][1] = 0;
        J[4][2] = 0;
        J[4][3] = 0;
        J[4][4] = sin(q5) / sqrt(sin(q5)*sin(q5));
        J[4][5] = 0;

        J[5][0] = 0;
        J[5][1] = 0;
        J[5][2] = 0;
        J[5][3] = 0;
        J[5][4] = 0;
        J[5][5] = 1;

}


//  CALCULA JACOBIANO CRUZ.
void CalculaJC() {

//////////////////////////      CALCULO DE LA MATRIZ TRASPUESTA
//////////////////////////

        for (i = 0; i<6; i++)
        {
                for (j = 0; j<6; j++)
                {
                        Jt[j][i] = J[i][j];
                }
        }
```

```
////////////////////////////      CALCULO DE LA MATRIZ J*J'
////////////////////////////

        Matrix.Multiply((float*)J, (float*)Jt, 6, 6, 6, (float*)JJ);

////////////////////////      CALCULO DE LA MATRIZ +I*Kp
//////////////////////////////

        Matrix.Add((float *)JJ, (float *)IK, 6, 6, (float *)JJJ);

////////////////////////      CALCULO DE LA MATRIZ INVERSA
//////////////////////////////

        // Primero es necesario copiarla para no reemplazarla

        for (i = 0; i<6; i++) {

                for (j = 0; j<6; j++) {

                        JJinv[i][j] = JJJ[i][j];
                }
        }

        Matrix.Invert((float*)JJinv, 6);

////////////////////////      CALCULO DE LA MATRIZ JCruz
//////////////////////////////////

        Matrix.Multiply((float*)Jt, (float*)JJinv, 6, 6, 6, (float*)Jc);

////////////////////////////      CAMBIAMOS EL SIGNO A J cruz
////////////////////////////


        for (i = 0; i < 6; i++)
        {
                for (j = 0; j < 6; j++)
                {
                        Jcn[i][j] = (-1)*Jc[i][j];
                }
        }

}


// MULTIPLICA JC POR ERRORKP Y CALCULA VEL EN RAD/S, RPM Y COORD MOTOR.
void CalculaVelDestino() {

        ////////////////////////////      CALCULO DE LAs VELOCIDADES
////////////////////////////

        Matrix.Multiply((float*)Jc, (float*)errorkpi, 6, 6, 1, (float*)qd);

        for (i = 0; i<6; i++)
        {
                if (qd[i][0] >= 0) { qds[i][0] = 1; } // Signo positivo.
                if (qd[i][0]<0) { qds[i][0] = -1; } // Signo negativo.
        }

////////////////////////      CALCULO DE LAs VELOCIDADES EN RPM
////////////////////////////

        for (i = 0; i < 6; i++)
        {
                qdr[i][0] = abs(qd[i][0] * 9.5493);
        }
```

```
//////////////////////      CALCULO DE LAs VELOCIDADES EN MOTORES
////////////////////////////

        for (i = 0; i < 6; i++)
        {
                qdm[i][0] = qdr[i][0] * 9;
                if (qdm[i][0] >= velmax[i][0]) { qdm[i][0] = velmax[i][0]; }  //
Podemos poner una variable de saturacion mejor.
        }

//////////////////      CALCULO DE LAs VELOCIDADES EN RPM  FINALES
////////////////////////////

        for (i = 0; i < 6; i++)
        {
                qdr[i][0] = qdm[i][0] * 0.1111;
        }

//////////////////      CALCULO DE LAs VELOCIDADES FINALES EN RAD/S
////////////////////////

        for (i = 0; i < 6; i++)
        {
                qdf[i][0] = qdr[i][0] * 0.1047;
        }



}

// CON LA VELOCIDAD ANTERIOR CALCULA LA POS DE DESTINO.
void CalculaPosDestino() {

        ////////////////////////      CALCULO DE LAS POSICIONES DESTINO
////////////////////////
        incq1 = qds[0][0] * qdf[0][0] * t;
        incq2 = qds[1][0] * qdf[1][0] * t;
        incq3 = qds[2][0] * qdf[2][0] * t;
        incq4 = qds[3][0] * qdf[3][0] * t;
        incq5 = qds[4][0] * qdf[4][0] * t;
        incq6 = qds[5][0] * qdf[5][0] * t;

        q1des = q1 + (2* incq1);
        q2des = q2 + (2* incq2);
        q3des = q3 + (2* incq3);
        q4des = q4 + (2* incq4);
        q5des = q5 + (2* incq5);
        q6des = q6 + (2* incq6);

        q1mdes = q1des * 196.0784 + 512;
        q2mdes = q2des * 196.0784 + 512;
        q3mdes = q3des * 196.0784 + 512;
        q4mdes = q4des * 196.0784 + 512;
        q5mdes = ((-1)*q5des*196.0784) + 512;
        q6mdes = q6des * 196.0784 + 512;
}


// FUNCION PARA COLOCAR LOS VALORES CORRECTOS EN LOS VECTORES QUE SE USAN PARA ACTUAL
EN LOS MOTORES.
void CalculaActuacionMotores() {

////////////////////////////////////////////////////////////////////////////////////
////////////
////////////////////////////      ACTUACION SOBRE LOS MOTORES
////////////////////////////
////////////////////////////////////////////////////////////////////////////////////
////////////
```

```
//------------    ARTICULACION 1 ( 1 MOTOR )    --------------------------//
        vectpos[0][1] = q1mdes;
        vectvel[0][1] = qdm[0][0]

//------------    ARTICULACION 2 ( 2 MOTORES )    ----------------------------//

        vectvel[1][1] = qdm[1][0];
        vectvel[2][1] = qdm[1][0];

        if (q2mdes<q2m)
        {

                vectpos[1][1] = q2mdes;
                vectpos[2][1] = q22m + (q2m - q2mdes);

        }

        if (q2mdes>q2m)
        {

                vectpos[1][1] = q2mdes;
                vectpos[2][1] = q22m - (q2mdes - q2m);

        }

//------------    ARTICULACION 3 ( 2 MOTORES )    -------------------------//

        vectvel[3][1] = qdm[2][0];
        vectvel[4][1] = qdm[2][0];

        if (q3mdes<q3m)
        {
                vectpos[3][1] = q3mdes;
                vectpos[4][1] = q33m + (q3m - q3mdes);
        }

        if (q3mdes>q3m)
        {
                vectpos[3][1] = q3mdes;
                vectpos[4][1] = q33m - (q3mdes - q3m);
        }

//------------    ARTICULACION 4 ( 1 MOTOR )    --------------------------//

        vectpos[5][1] = q4mdes;
        vectvel[5][1] = qdm[3][0];

//------------    ARTICULACION 5 ( 1 MOTOR )    --------------------------//

        vectpos[6][1] = q5mdes;
        vectvel[6][1] = qdm[4][0];
//------------    ARTICULACION 6 ( 1 MOTOR )    --------------------------//


        vectpos[7][1] = q6mdes;
        vectvel[7][1] = qdm[5][0];

}



// FUNCION QUE MANDA LA SEÑAL A LOS MOTORES.
void Actua() {

        ////////////////////////      VELOCIDAD DESEADA A LOS MOTORES
////////////////////////////

        dxlSyncWrite(vectvel, 8, AX_GOAL_SPEED_L, 2);
```

```
        ////////////////////////      POSICION DESEADA A LOS MOTORES
//////////////////////////

        dxlSyncWrite(vectpos, 8, AX_GOAL_POSITION_L, 2);



}



void Leer()
{
        while (esperaleer == 0)
        {

                if (Serial.available()) {
                        //Lectura de caracteres
                        InCadena = Serial.read();
                        //Suma de caracteres en variable string
                        cadena += InCadena;

                        //Imprime la variable con los caracteres acumulados hasta la
","
                        if (InCadena == '.') {
                                Serial.print("Has introducido: ");
                                Serial.println(cadena);
                                //Borra la variable string para almacenar nuevos
datos
                                cadenaleida = cadena;
                                cadena = "";
                                esperaleer = 1;
                        }
                }
        }
}


///////////////// Calcula Damping /////////////

void CalculaIK()
{
  ex1=cos(q4)*cos(q4)*sin(q5)*sin(q5)*sin(q5) +
sin(q4)*sin(q4)*sin(q5)*sin(q5)*sin(q5) + cos(q4)*cos(q4)*cos(q5)*cos(q5)*sin(q5) +
cos(q5)*cos(q5)*sin(q4)*sin(q4)*sin(q5);

  det=(0.00000017*(ex1)*(4299.0*sin(q1 + q2)*sin(q1 + q2)*sin(q2 + q3)*sin(q3) +
4299.0*sin(q2 + q3)*cos(q1 - 1.0*q2)*cos(q1 - 1.0*q2)*sin(q3) + 4299.0*sin(q2 +
q3)*sin(q1 - 1.0*q2)*sin(q1 - 1.0*q2)*sin(q3) + 5611.0*cos(q1 + q2)*cos(q1 +
q2)*sin(q2)*sin(q3) + 5611.0*sin(q1 + q2)*sin(q1 + q2)*sin(q2)*sin(q3) + 5611.0*cos(q1
- 1.0*q2)*cos(q1 - 1.0*q2)*sin(q2)*sin(q3) + 5611.0*sin(q1 - 1.0*q2)*sin(q1 -
1.0*q2)*sin(q2)*sin(q3) - 1.1e4*cos(q1 + q2)*sin(q2 + q3)*sin(q1 - 1.0*q2) +
1.1e4*sin(q1 + q2)*sin(q2 + q3)*cos(q1 - 1.0*q2) + 4299.0*cos(q1 + q2)*cos(q1 +
q2)*sin(q2 + q3)*sin(q3) - 8711.0*cos(q1 + q2)*sin(q1 -
1.0*q2)*sin(q2)*sin(q3)*sin(q3) + 8711.0*sin(q1 + q2)*cos(q1 -
1.0*q2)*sin(q2)*sin(q3)*sin(q3) + 4955.0*sin(q2 + q3)*sin(q1 - 1.0*q2)*sin(q1 -
1.0*q2)*cos(q5)*sin(q3) + 6388.0*cos(q1 + q2 + q3)*sin(q1 - 1.0*q2)*cos(q5)*sin(q2) -
6388.0*sin(q1 + q2 + q3)*cos(q1 - 1.0*q2)*cos(q5)*sin(q2) - 6388.0*cos(q2 - 1.0*q1 +
q3)*sin(q1 + q2)*cos(q5)*sin(q2) - 6388.0*sin(q2 - 1.0*q1 + q3)*cos(q1 +
q2)*cos(q5)*sin(q2) - 5622.0*cos(q1 + q2 + q3)*sin(q1 + q2)*sin(q2 +
q3)*cos(q5)*cos(q5) + 5622.0*sin(q1 + q2 + q3)*cos(q1 + q2)*sin(q2 +
q3)*cos(q5)*cos(q5) + 6388.0*cos(q2 - 1.0*q1 + q3)*sin(q1 - 1.0*q2)*cos(q5)*sin(q2) +
6388.0*sin(q2 - 1.0*q1 + q3)*cos(q1 - 1.0*q2)*cos(q5)*sin(q2) - 5622.0*cos(q1 + q2 +
q3)*sin(q2 + q3)*sin(q1 - 1.0*q2)*cos(q5)*cos(q5) + 5622.0*sin(q1 + q2 + q3)*sin(q2 +
q3)*cos(q1 - 1.0*q2)*cos(q5)*cos(q5) + 5622.0*cos(q2 - 1.0*q1 + q3)*sin(q1 +
q2)*sin(q2 + q3)*cos(q5)*cos(q5) + 5622.0*sin(q2 - 1.0*q1 + q3)*cos(q1 + q2)*sin(q2 +
q3)*cos(q5)*cos(q5) + 5622.0*cos(q2 - 1.0*q1 + q3)*sin(q2 + q3)*sin(q1 -
1.0*q2)*cos(q5)*cos(q5) + 5622.0*sin(q2 - 1.0*q1 + q3)*sin(q2 + q3)*cos(q1 -
```

```
1.0*q2)*cos(q5)*cos(q5) + 8588.0*cos(q1 + q2)*sin(q2 + q3)*cos(q1 - 1.0*q2)*sin(q3) -
8588.0*cos(q1 + q2)*sin(q2 + q3)*sin(q1 - 1.0*q2)*cos(q3) + 8588.0*sin(q1 + q2)*sin(q2
+ q3)*cos(q1 - 1.0*q2)*cos(q3) - 1.28e4*cos(q1 + q2)*sin(q2 + q3)*sin(q1 -
1.0*q2)*cos(q5) + 1.28e4*sin(q1 + q2)*sin(q2 + q3)*cos(q1 - 1.0*q2)*cos(q5) +
8588.0*sin(q1 + q2)*sin(q2 + q3)*sin(q1 - 1.0*q2)*sin(q3) - 1.12e4*cos(q1 + q2)*cos(q1
- 1.0*q2)*sin(q2)*sin(q3) + 1.12e4*cos(q1 + q2)*sin(q1 - 1.0*q2)*cos(q3)*sin(q2) -
1.12e4*sin(q1 + q2)*cos(q1 - 1.0*q2)*cos(q3)*sin(q2) - 1.12e4*sin(q1 + q2)*sin(q1 -
1.0*q2)*sin(q2)*sin(q3) - 4888.0*cos(q1 + q2 + q3)*sin(q1 + q2)*sin(q2 + q3)*cos(q5) +
4888.0*sin(q1 + q2 + q3)*cos(q1 + q2)*sin(q2 + q3)*cos(q5) + 4955.0*cos(q1 +
q2)*cos(q1 + q2)*sin(q2 + q3)*cos(q5)*sin(q3) - 4888.0*cos(q1 + q2 + q3)*sin(q2 +
q3)*sin(q1 - 1.0*q2)*cos(q5) + 4888.0*sin(q1 + q2 + q3)*sin(q2 + q3)*cos(q1 -
1.0*q2)*cos(q5) + 4888.0*cos(q2 - 1.0*q1 + q3)*sin(q1 + q2)*sin(q2 + q3)*cos(q5) +
4888.0*sin(q2 - 1.0*q1 + q3)*cos(q1 + q2)*sin(q2 + q3)*cos(q5) - 1.12e4*cos(q1 + q2 +
q3)*sin(q2 - 1.0*q1 + q3)*cos(q5)*cos(q5)*sin(q2) - 1.12e4*sin(q1 + q2 + q3)*cos(q2 -
1.0*q1 + q3)*cos(q5)*cos(q5)*sin(q2) + 4955.0*sin(q1 + q2)*sin(q1 + q2)*sin(q2 +
q3)*cos(q5)*sin(q3) - 6388.0*cos(q1 + q2 + q3)*sin(q1 + q2)*cos(q5)*sin(q2) +
6388.0*sin(q1 + q2 + q3)*cos(q1 + q2)*cos(q5)*sin(q2) + 8711.0*cos(q1 + q2)*sin(q1 -
1.0*q2)*cos(q3)*cos(q3)*sin(q2) - 8711.0*sin(q1 + q2)*cos(q1 -
1.0*q2)*cos(q3)*cos(q3)*sin(q2) + 4955.0*sin(q2 + q3)*cos(q1 - 1.0*q2)*cos(q1 -
1.0*q2)*cos(q5)*sin(q3) + 4888.0*cos(q2 - 1.0*q1 + q3)*sin(q2 + q3)*sin(q1 -
1.0*q2)*cos(q5) + 4888.0*sin(q2 - 1.0*q1 + q3)*sin(q2 + q3)*cos(q1 - 1.0*q2)*cos(q5) -
6388.0*sin(q2 - 1.0*q1 + q3)*sin(q1 - 1.0*q2)*cos(q4)*sin(q2)*sin(q5) + 1.12e4*cos(q1
+ q2 + q3)*sin(q2 - 1.0*q1 + q3)*cos(q4)*cos(q4)*sin(q2)*sin(q5)*sin(q5) +
1.12e4*sin(q1 + q2 + q3)*cos(q2 - 1.0*q1 + q3)*cos(q4)*cos(q4)*sin(q2)*sin(q5)*sin(q5)
+ 9900.0*cos(q1 + q2)*sin(q2 + q3)*cos(q1 - 1.0*q2)*cos(q5)*sin(q3) - 9900.0*cos(q1 +
q2)*sin(q2 + q3)*sin(q1 - 1.0*q2)*cos(q3)*cos(q5) + 9900.0*sin(q1 + q2)*sin(q2 +
q3)*cos(q1 - 1.0*q2)*cos(q3)*cos(q5) - 1.28e4*cos(q1 + q2)*cos(q2 + q3)*sin(q1 -
1.0*q2)*cos(q4)*sin(q5) + 1.28e4*cos(q2 + q3)*sin(q1 + q2)*cos(q1 -
1.0*q2)*cos(q4)*sin(q5) + 5622.0*cos(q1 + q2 + q3)*cos(q1 + q2)*cos(q2 +
q3)*cos(q4)*cos(q4)*sin(q5)*sin(q5) + 5622.0*cos(q1 + q2 + q3)*cos(q2 + q3)*cos(q1 -
1.0*q2)*cos(q4)*cos(q4)*sin(q5)*sin(q5) + 5622.0*cos(q2 - 1.0*q1 + q3)*cos(q1 +
q2)*cos(q2 + q3)*cos(q4)*cos(q4)*sin(q5)*sin(q5) + 9900.0*sin(q1 + q2)*sin(q2 +
q3)*sin(q1 - 1.0*q2)*cos(q5)*sin(q3) + 5622.0*sin(q1 + q2 + q3)*cos(q2 + q3)*sin(q1 +
q2)*cos(q4)*cos(q4)*sin(q5)*sin(q5) - 1.74e4*cos(q1 + q2)*cos(q1 -
1.0*q2)*cos(q3)*sin(q2)*sin(q3) - 9755.0*cos(q1 + q2)*sin(q2 +
q3)*sin(q1)*sin(q4)*sin(q5) + 9755.0*sin(q1 + q2)*sin(q2 + q3)*cos(q1)*sin(q4)*sin(q5)
+ 5622.0*cos(q2 - 1.0*q1 + q3)*cos(q2 + q3)*cos(q1 -
1.0*q2)*cos(q4)*cos(q4)*sin(q5)*sin(q5) + 5622.0*sin(q1 + q2 + q3)*cos(q2 + q3)*sin(q1
- 1.0*q2)*cos(q4)*cos(q4)*sin(q5)*sin(q5) - 5622.0*sin(q2 - 1.0*q1 + q3)*cos(q2 +
q3)*sin(q1 + q2)*cos(q4)*cos(q4)*sin(q5)*sin(q5) - 1.74e4*sin(q1 + q2)*sin(q1 -
1.0*q2)*cos(q3)*sin(q2)*sin(q3) - 9755.0*sin(q2 + q3)*cos(q1 -
1.0*q2)*sin(q1)*sin(q4)*sin(q5) + 9755.0*sin(q2 + q3)*sin(q1 -
1.0*q2)*cos(q1)*sin(q4)*sin(q5) + 4888.0*cos(q1 + q2 + q3)*cos(q1 + q2)*sin(q2 +
q3)*cos(q4)*sin(q5) - 5622.0*sin(q2 - 1.0*q1 + q3)*cos(q2 + q3)*sin(q1 -
1.0*q2)*cos(q4)*cos(q4)*sin(q5)*sin(q5) + 4955.0*cos(q1 + q2)*cos(q1 + q2)*cos(q2 +
q3)*cos(q4)*sin(q3)*sin(q5) + 4888.0*cos(q1 + q2 + q3)*sin(q2 + q3)*cos(q1 -
1.0*q2)*cos(q4)*sin(q5) + 4888.0*cos(q2 - 1.0*q1 + q3)*cos(q1 + q2)*sin(q2 +
q3)*cos(q4)*sin(q5) + 4888.0*sin(q1 + q2 + q3)*sin(q1 + q2)*sin(q2 +
q3)*cos(q4)*sin(q5) + 4955.0*cos(q2 + q3)*sin(q1 + q2)*sin(q1 +
q2)*cos(q4)*sin(q3)*sin(q5) + 6388.0*cos(q1 + q2 + q3)*cos(q1 +
q2)*cos(q4)*sin(q2)*sin(q5) + 4955.0*cos(q2 + q3)*cos(q1 - 1.0*q2)*cos(q1 -
1.0*q2)*cos(q4)*sin(q3)*sin(q5) + 4888.0*cos(q2 - 1.0*q1 + q3)*sin(q2 + q3)*cos(q1 -
1.0*q2)*cos(q4)*sin(q5) + 4888.0*sin(q1 + q2 + q3)*sin(q2 + q3)*sin(q1 -
1.0*q2)*cos(q4)*sin(q5) - 4888.0*sin(q2 - 1.0*q1 + q3)*sin(q1 + q2)*sin(q2 +
q3)*cos(q4)*sin(q5) + 4955.0*cos(q2 + q3)*sin(q1 - 1.0*q2)*sin(q1 -
1.0*q2)*cos(q4)*sin(q3)*sin(q5) - 9900.0*cos(q1 + q2 + q3)*cos(q1 -
1.0*q2)*cos(q5)*sin(q2)*sin(q3) + 9900.0*cos(q1 + q2 + q3)*sin(q1 -
1.0*q2)*cos(q3)*cos(q5)*sin(q2) - 9900.0*sin(q1 + q2 + q3)*cos(q1 -
1.0*q2)*cos(q3)*cos(q5)*sin(q2) - 9900.0*cos(q2 - 1.0*q1 + q3)*cos(q1 +
q2)*cos(q5)*sin(q2)*sin(q3) - 9900.0*cos(q2 - 1.0*q1 + q3)*sin(q1 +
q2)*cos(q3)*cos(q5)*sin(q2) - 9900.0*sin(q2 - 1.0*q1 + q3)*cos(q1 +
q2)*cos(q3)*cos(q5)*sin(q2) - 6388.0*cos(q1 + q2 + q3)*cos(q1 -
1.0*q2)*cos(q4)*sin(q2)*sin(q5) - 6388.0*cos(q2 - 1.0*q1 + q3)*cos(q1 +
q2)*cos(q4)*sin(q2)*sin(q5) + 6388.0*sin(q1 + q2 + q3)*sin(q1 +
q2)*cos(q4)*sin(q2)*sin(q5) - 4888.0*sin(q2 - 1.0*q1 + q3)*sin(q2 + q3)*sin(q1 -
1.0*q2)*cos(q4)*sin(q5) + 6388.0*cos(q2 - 1.0*q1 + q3)*cos(q1 -
1.0*q2)*cos(q4)*sin(q2)*sin(q5) - 9900.0*sin(q1 + q2 + q3)*sin(q1 -
1.0*q2)*cos(q5)*sin(q2)*sin(q3) + 9900.0*sin(q2 - 1.0*q1 + q3)*sin(q1 +
q2)*cos(q5)*sin(q2)*sin(q3) - 6388.0*sin(q1 + q2 + q3)*sin(q1 -
```

```
1.0*q2)*cos(q4)*sin(q2)*sin(q5) + 6388.0*sin(q2 - 1.0*q1 + q3)*sin(q1 +
q2)*cos(q4)*sin(q2)*sin(q5) + 9900.0*cos(q1 + q2)*cos(q2 + q3)*cos(q1 -
1.0*q2)*cos(q4)*sin(q3)*sin(q5) - 9900.0*cos(q1 + q2)*cos(q2 + q3)*sin(q1 -
1.0*q2)*cos(q3)*cos(q4)*sin(q5) + 9900.0*cos(q2 + q3)*sin(q1 + q2)*cos(q1 -
1.0*q2)*cos(q3)*cos(q4)*sin(q5) - 2.25e4*cos(q1 + q2 + q3)*cos(q2 - 1.0*q1 +
q3)*cos(q4)*cos(q5)*sin(q2)*sin(q5) + 9900.0*cos(q2 + q3)*sin(q1 + q2)*sin(q1 -
1.0*q2)*cos(q4)*sin(q3)*sin(q5) - 1.12e4*cos(q1 + q2 +
q3)*cos(q1)*cos(q4)*sin(q2)*sin(q4)*sin(q5)*sin(q5) - 1.12e4*cos(q1 + q2)*sin(q2 +
q3)*cos(q5)*sin(q1)*sin(q4)*sin(q5) + 1.12e4*sin(q1 + q2)*sin(q2 +
q3)*cos(q1)*cos(q5)*sin(q4)*sin(q5) + 2.25e4*sin(q1 + q2 + q3)*sin(q2 - 1.0*q1 +
q3)*cos(q4)*cos(q5)*sin(q2)*sin(q5) + 1.12e4*cos(q2 - 1.0*q1 +
q3)*cos(q1)*cos(q4)*sin(q2)*sin(q4)*sin(q5)*sin(q5) - 1.12e4*sin(q1 + q2 +
q3)*cos(q4)*sin(q1)*sin(q2)*sin(q4)*sin(q5)*sin(q5) - 1.12e4*sin(q2 + q3)*cos(q1 -
1.0*q2)*cos(q5)*sin(q1)*sin(q4)*sin(q5) + 1.12e4*sin(q2 + q3)*sin(q1 -
1.0*q2)*cos(q1)*cos(q5)*sin(q4)*sin(q5) + 5622.0*cos(q1 + q2 + q3)*cos(q1 + q2)*sin(q2
+ q3)*cos(q4)*cos(q5)*sin(q5) - 5622.0*cos(q1 + q2 + q3)*cos(q2 + q3)*sin(q1 +
q2)*cos(q4)*cos(q5)*sin(q5) + 5622.0*sin(q1 + q2 + q3)*cos(q1 + q2)*cos(q2 +
q3)*cos(q4)*cos(q5)*sin(q5) - 1.12e4*sin(q2 - 1.0*q1 +
q3)*cos(q4)*sin(q1)*sin(q2)*sin(q4)*sin(q5)*sin(q5) - 9900.0*cos(q1 +
q2)*cos(q1)*sin(q2)*sin(q3)*sin(q4)*sin(q5) + 9900.0*cos(q1 +
q2)*cos(q3)*sin(q1)*sin(q2)*sin(q4)*sin(q5) - 9900.0*sin(q1 +
q2)*cos(q1)*cos(q3)*sin(q2)*sin(q4)*sin(q5) - 5622.0*cos(q1 + q2 + q3)*cos(q2 +
q3)*sin(q1 - 1.0*q2)*cos(q4)*cos(q5)*sin(q5) + 5622.0*cos(q1 + q2 + q3)*sin(q2 +
q3)*cos(q1 - 1.0*q2)*cos(q4)*cos(q5)*sin(q5) + 5622.0*sin(q1 + q2 + q3)*cos(q2 +
q3)*cos(q1 - 1.0*q2)*cos(q4)*cos(q5)*sin(q5) + 5622.0*cos(q2 - 1.0*q1 + q3)*cos(q1 +
q2)*sin(q2 + q3)*cos(q4)*cos(q5)*sin(q5) + 5622.0*cos(q2 - 1.0*q1 + q3)*cos(q2 +
q3)*sin(q1 + q2)*cos(q4)*cos(q5)*sin(q5) + 5622.0*sin(q2 - 1.0*q1 + q3)*cos(q1 +
q2)*cos(q2 + q3)*cos(q4)*cos(q5)*sin(q5) + 5622.0*sin(q1 + q2 + q3)*sin(q1 +
q2)*sin(q2 + q3)*cos(q4)*cos(q5)*sin(q5) + 9900.0*cos(q1 -
1.0*q2)*cos(q1)*sin(q2)*sin(q3)*sin(q4)*sin(q5) + 9900.0*cos(q1 -
1.0*q2)*cos(q3)*sin(q1)*sin(q2)*sin(q4)*sin(q5) - 9900.0*sin(q1 -
1.0*q2)*cos(q1)*cos(q3)*sin(q2)*sin(q4)*sin(q5) - 1.12e4*cos(q1 + q2)*cos(q2 +
q3)*cos(q4)*sin(q1)*sin(q4)*sin(q5)*sin(q5) + 1.12e4*cos(q2 + q3)*sin(q1 +
q2)*cos(q1)*cos(q4)*sin(q4)*sin(q5)*sin(q5) - 9900.0*sin(q1 +
q2)*sin(q1)*sin(q2)*sin(q3)*sin(q4)*sin(q5) + 5622.0*cos(q2 - 1.0*q1 + q3)*cos(q2 +
q3)*sin(q1 - 1.0*q2)*cos(q4)*cos(q5)*sin(q5) + 5622.0*cos(q2 - 1.0*q1 + q3)*sin(q2 +
q3)*cos(q1 - 1.0*q2)*cos(q4)*cos(q5)*sin(q5) + 5622.0*sin(q2 - 1.0*q1 + q3)*cos(q2 +
q3)*cos(q1 - 1.0*q2)*cos(q4)*cos(q5)*sin(q5) + 5622.0*sin(q1 + q2 + q3)*sin(q2 +
q3)*sin(q1 - 1.0*q2)*cos(q4)*cos(q5)*sin(q5) - 5622.0*sin(q2 - 1.0*q1 + q3)*sin(q1 +
q2)*sin(q2 + q3)*cos(q4)*cos(q5)*sin(q5) - 1.12e4*cos(q2 + q3)*cos(q1 -
1.0*q2)*cos(q4)*sin(q1)*sin(q4)*sin(q5)*sin(q5) + 1.12e4*cos(q2 + q3)*sin(q1 -
1.0*q2)*cos(q1)*cos(q4)*sin(q4)*sin(q5)*sin(q5) + 9900.0*sin(q1 -
1.0*q2)*sin(q1)*sin(q2)*sin(q3)*sin(q4)*sin(q5) - 9900.0*cos(q1 + q2 + q3)*cos(q1 -
1.0*q2)*cos(q3)*cos(q4)*sin(q2)*sin(q5) - 9900.0*cos(q2 - 1.0*q1 + q3)*cos(q1 +
q2)*cos(q3)*cos(q4)*sin(q2)*sin(q5) - 5622.0*sin(q2 - 1.0*q1 + q3)*sin(q2 + q3)*sin(q1
- 1.0*q2)*cos(q4)*cos(q5)*sin(q5) - 9900.0*cos(q1 + q2 + q3)*sin(q1 -
1.0*q2)*cos(q4)*sin(q2)*sin(q3)*sin(q5) + 9900.0*sin(q1 + q2 + q3)*cos(q1 -
1.0*q2)*cos(q4)*sin(q2)*sin(q3)*sin(q5) - 9900.0*sin(q1 + q2 + q3)*sin(q1 -
1.0*q2)*cos(q3)*cos(q4)*sin(q2)*sin(q5) + 9900.0*cos(q2 - 1.0*q1 + q3)*sin(q1 +
q2)*cos(q4)*sin(q2)*sin(q3)*sin(q5) + 9900.0*sin(q2 - 1.0*q1 + q3)*cos(q1 +
q2)*cos(q4)*sin(q2)*sin(q3)*sin(q5) + 9900.0*sin(q2 - 1.0*q1 + q3)*sin(q1 +
q2)*cos(q3)*cos(q4)*sin(q2)*sin(q5) + 1.12e4*cos(q1 + q2 +
q3)*cos(q5)*sin(q1)*sin(q2)*sin(q4)*sin(q5) - 1.12e4*sin(q1 + q2 +
q3)*cos(q1)*cos(q5)*sin(q2)*sin(q4)*sin(q5) + 1.12e4*cos(q2 - 1.0*q1 +
q3)*cos(q5)*sin(q1)*sin(q2)*sin(q4)*sin(q5) + 1.12e4*sin(q2 - 1.0*q1 +
q3)*cos(q1)*cos(q5)*sin(q2)*sin(q4)*sin(q5)))/(sqrt(cos(q4)*cos(q4)*sin(q5)*sin(q5) +
sin(q4)*sin(q4)*sin(q5)*sin(q5))*(cos(q5)*cos(q5) + cos(q4)*cos(q4)*sin(q5)*sin(q5) +
sin(q4)*sin(q4)*sin(q5)*sin(q5)));

  k=k0*exp(-(det*det)/(2*eps));

  for (i=0;i<6;i++)
  {
      IK[i][i]=k;

  }

}
```

7.6 Anexo VI: Código Python Para Reconocimiento de Ángulos:

```
import cv2
import numpy as np
import math
import time

#Iniciamos la camara
captura = cv2.VideoCapture(0)


#Establecemos el rango de colores que vamos a detectar

# Color Cyan
B1=np.array([100,130,85])
A1=np.array([110,155,100])
x1a = 1
y1a = 1
x1 = 1
y1 = 1

# Color Morado
B2=np.array([117,95,90])
A2=np.array([140,110,120])
x2a = 1
y2a = 1
x2 = 1
y2 = 1
# Color Verde
B3=np.array([80,60,60])
A3=np.array([110,85,95])
x3a = 1
y3a = 1
x3 = 1
y3 = 1

# Color Amarillo
B4=np.array([30,70,90])
A4=np.array([50,125,135])
x4a = 1
y4a = 1
x4 = 1
y4 = 1

# Color Rojo
B5=np.array([140,100,50])
A5=np.array([170,130,95])
x5a = 1
y5a = 1
x5 = 1
y5 = 1

# Color Azul
B6=np.array([113,155,85])
A6=np.array([125,185,115])
x6a = 1
y6a = 1
x6 = 1
y6 = 1

# Color Crema
B7=np.array([0,60,60])
A7=np.array([20,120,140])
x7a = 1
y7a = 1
x7 = 1
y7 = 1
```

```
Flag=1


_, imagen = captura.read()
hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)

time.sleep(3)



while(1):

    #Capturamos una imagen y la convertimos de RGB -> HSV
    _, imagen = captura.read()
    hsv = cv2.cvtColor(imagen, cv2.COLOR_BGR2HSV)


    time.sleep(5)


    Result=np.zeros((int(captura.get(4)),int(captura.get(3)),3),dtype=np.uint8)
    Datos=np.zeros((int(captura.get(4)),int(captura.get(3)),3),dtype=np.uint8)


    #Crear una mascara con solo los pixeles dentro del rango
    m1 = cv2.inRange(hsv, B1, A1)
    m2 = cv2.inRange(hsv, B2, A2)
    m3 = cv2.inRange(hsv, B3, A3)
    m4 = cv2.inRange(hsv, B4, A4)
    m5 = cv2.inRange(hsv, B5, A5)
    m6 = cv2.inRange(hsv, B6, A6)
    m7 = cv2.inRange(hsv, B7, A7)

    #Filtrar el ruido con un CLOSE/OPEN
    kernel = np.ones((10,10),np.uint8)
    m1 = cv2.morphologyEx(m1, cv2.MORPH_CLOSE, kernel)
    m1 = cv2.morphologyEx(m1, cv2.MORPH_OPEN, kernel)
    m2 = cv2.morphologyEx(m2, cv2.MORPH_CLOSE, kernel)
    m2 = cv2.morphologyEx(m2, cv2.MORPH_OPEN, kernel)
    m3 = cv2.morphologyEx(m3, cv2.MORPH_CLOSE, kernel)
    m3 = cv2.morphologyEx(m3, cv2.MORPH_OPEN, kernel)
    m4 = cv2.morphologyEx(m4, cv2.MORPH_CLOSE, kernel)
    m4 = cv2.morphologyEx(m4, cv2.MORPH_OPEN, kernel)
    m5 = cv2.morphologyEx(m5, cv2.MORPH_CLOSE, kernel)
    m5 = cv2.morphologyEx(m5, cv2.MORPH_OPEN, kernel)
    m6 = cv2.morphologyEx(m6, cv2.MORPH_CLOSE, kernel)
    m6 = cv2.morphologyEx(m6, cv2.MORPH_OPEN, kernel)
    m7 = cv2.morphologyEx(m7, cv2.MORPH_CLOSE, kernel)
    m7 = cv2.morphologyEx(m7, cv2.MORPH_OPEN, kernel)



#Encontrar el area de los objetos que detecta la camara
    mom1 = cv2.moments(m1)
    area1 = mom1['m00']
    mom2 = cv2.moments(m2)
    area2 = mom2['m00']
    mom3 = cv2.moments(m3)
    area3 = mom3['m00']
    mom4 = cv2.moments(m4)
    area4 = mom4['m00']
    mom5 = cv2.moments(m5)
    area5 = mom5['m00']
    mom6 = cv2.moments(m6)
    area6 = mom6['m00']
    mom7 = cv2.moments(m7)
    area7 = mom7['m00']
```

```
    #Buscamos el centro x, y del objeto
if(area1 > 80000):

    x1a = int(mom1['m10']/mom1['m00'])
    y1a = int(mom1['m01']/mom1['m00'])

if(area2 > 80000):

    x2a = int(mom2['m10']/mom2['m00'])
    y2a = int(mom2['m01']/mom2['m00'])

if(area3 > 80000):

    x3a = int(mom3['m10']/mom3['m00'])
    y3a = int(mom3['m01']/mom3['m00'])

if(area4 > 80000):

    x4a = int(mom4['m10']/mom4['m00'])
    y4a = int(mom4['m01']/mom4['m00'])

if(area5 > 80000):

    x5a = int(mom5['m10']/mom5['m00'])
    y5a = int(mom5['m01']/mom5['m00'])

if(area6 > 80000):

    x6a = int(mom6['m10']/mom6['m00'])
    y6a = int(mom6['m01']/mom6['m00'])

if(area7 > 80000):

    x7a = int(mom7['m10']/mom7['m00'])
    y7a = int(mom7['m01']/mom7['m00'])

    # Ajustamos centro para color 1

if ( abs(x1a-x1)>5 ):

    x1=x1a

else:

    x1= int( (x1a+x1)/2 )

if ( abs(y1a-y1)>5 ):

    y1=y1a

else:

    y1= int( (y1a+y1)/2 )

# Ajustamos centro para color 2

if ( abs(x2a-x2)>5 ):

    x2=x2a

else:

    x2= int( (x2a+x2)/2 )

if ( abs(y2a-y2)>5 ):

    y2=y2a

else:
```

```
    y2= int( (y2a+y2)/2 )
# Ajustamos centro para color 3
if ( abs(x3a-x3)>5 ):
    x3=x3a
else:
    x3= int( (x3a+x3)/2 )
if ( abs(y3a-y3)>5 ):
    y3=y3a
else:
    y3= int( (y3a+y3)/2 )
# Ajustamos centro para color 4
if ( abs(x4a-x4)>5 ):
    x4=x4a
else:
    x4= int( (x4a+x4)/2 )
if ( abs(y4a-y4)>5 ):
    y4=y4a
else:
    y4= int( (y4a+y4)/2 )
# Ajustamos centro para color 5
if ( abs(x5a-x5)>5 ):
    x5=x5a
else:
    x5= int( (x5a+x5)/2 )
if ( abs(y5a-y5)>5 ):
    y5=y5a
else:
    y5= int( (y5a+y5)/2 )
# Ajustamos centro para color 6
if ( abs(x6a-x6)>5 ):
    x6=x6a
else:
    x6= int( (x6a+x6)/2 )
if ( abs(y6a-y6)>5 ):
```

```
        y6=y6a

    else:

        y6= int( (y6a+y6)/2 )

    # Ajustamos centro para color 7

    if ( abs(x7a-x7)>5 ):

        x7=x7a

    else:

        x7= int( (x7a+x7)/2 )

    if ( abs(y7a-y7)>5 ):

        y7=y7a

    else:

        y7= int( (y7a+y7)/2 )

    ## CALCULAMOS EL TRIANGULO 1 FORMADO POR 1,2 Y 3:

    distancia21=math.sqrt(((x2-x1)**2)+((y2-y1)**2))
    if (distancia21<=0):
        distancia21=0.001
    print ("distancia21 = ", distancia21)

    distancia31=math.sqrt(((x3-x1)**2)+((y3-y1)**2))
    if (distancia31<=0):
        distancia31=0.001
    print ("distancia31 = ", distancia31)

    distancia32=math.sqrt(((x3-x2)**2)+((y3-y2)**2))
    if (distancia32<=0):
        distancia32=0.001
    print ("distancia32 = ", distancia32)

    #Calculamos los angulos:
    gamma1=math.acos(((distancia21**2)+(distancia32**2)-
(distanci31**2))/(2*distancia21*distancia32))


    ## CALCULAMOS EL TRIANGULO 1 FORMADO POR 3,4 Y 5:

    distancia43=math.sqrt(((x4-x3)**2)+((y4-y3)**2))
    if (distancia43<=0):
        distancia43=0.001
    print ("distancia43 = ", distancia43)

    distancia54=math.sqrt(((x5-x4)**2)+((y5-y4)**2))
    if (distancia54<=0):
        distancia54=0.001
    print ("distancia54 = ", distancia54)

    distancia53=math.sqrt(((x5-x3)**2)+((y5-y3)**2))
    if (distancia53<=0):
        distancia53=0.001
    print ("distancia53 = ", distancia53)

    #Calculamos los angulos:
    gamma2=math.acos(((distancia43**2)+(distancia54**2)-
(distanci53**2))/(2*distancia43*distancia54))


    ## CALCULAMOS EL TRIANGULO 1 FORMADO POR 5,6 Y 7:
```

```
    distancia65=math.sqrt(((x6-x5)**2)+((y6-y5)**2))
    if (distancia65<=0):
        distancia65=0.001
    print ("distancia65 = ", distancia65)

    distancia76=math.sqrt(((x7-x6)**2)+((y7-y6)**2))
    if (distancia76<=0):
        distancia76=0.001
    print ("distancia76 = ", distancia76)

    distancia75=math.sqrt(((x7-x5)**2)+((y7-y5)**2))
    if (distancia75<=0):
        distancia75=0.001
    print ("distancia75 = ", distancia75)

    #Calculamos los angulos:
    gamma3=math.acos(((distancia65**2)+(distancia76**2)-
(distanci75**2))/(2*distancia65*distancia76))


    #Mostramos sus coordenadas por pantalla
    print ("x1 = ", x1)
    print ("y1 = ", y1)
    print ("x2 = ", x2)
    print ("y2 = ", y2)
    print ("x3 = ", x3)
    print ("y3 = ", y3)
    print ("x4 = ", x4)
    print ("y4 = ", y4)
    print ("x5 = ", x5)
    print ("y5 = ", y5)
    print ("x6 = ", x6)
    print ("y6 = ", y6)
    print ("x7 = ", x7)
    print ("y7 = ", y7)
    print ("gamma1 = ", gamma1)
    print ("gamma2 = ", gamma2)
    print ("gamma3 = ", gamma3)

    #Dibujamos una marca en el centro del objeto
    cv2.circle(imagen, (x1, y1), 2,(204,204,0), -1)
    cv2.circle(Result, (x1, y1), 2,(204,204,0), -1)

    cv2.circle(imagen, (x2, y2), 2,(255,0,127), -1)
    cv2.circle(Result, (x2, y2), 2,(255,0,127), -1)

    cv2.circle(imagen, (x3, y3), 2,(0,255,0), -1)
    cv2.circle(Result, (x3, y3), 2,(0,255,0), -1)

    cv2.circle(imagen, (x4, y4), 2,(0,255,255), -1)
    cv2.circle(Result, (x4, y4), 2,(0,255,255), -1)

    cv2.circle(imagen, (x5, y5), 2,(0,0,255), -1)
    cv2.circle(Result, (x5, y5), 2,(0,0,255), -1)

    cv2.circle(imagen, (x6, y6), 2,(255,0,0), -1)
    cv2.circle(Result, (x6, y6), 2,(255,0,0), -1)

    cv2.circle(imagen, (x7, y7), 2,(0,128,255), -1)
    cv2.circle(Result, (x7, y7), 2,(0,128,255), -1)


    # Dibujamos las rectas que unen los  centros:
    cv2.line(imagen, (x3, y3), (x1, y1), (255,0,255),1)
    cv2.line(Result, (x3, y3), (x1, y1), (255,0,255),1)
    cv2.line(imagen, (x2, y2), (x1, y1), (255,0,255),1)
    cv2.line(Result, (x2, y2), (x1, y1), (255,0,255),1)
    cv2.line(imagen, (x3, y3), (x2, y2), (255,0,255),1)
```

```
    cv2.line(Result, (x3, y3), (x2, y2), (255,0,255),1)
    cv2.line(imagen, (xM1, yM1), (x2, y2), (255,255,255),1)
    cv2.line(Result, (xM1, yM1), (x2, y2), (255,255,255),1)

    cv2.line(imagen, (x5, y5), (x3, y3), (255,255,255),1)
    cv2.line(Result, (x5, y5), (x3, y3), (255,255,255),1)
    cv2.line(imagen, (x4, y4), (x3, y3), (255,255,255),1)
    cv2.line(Result, (x4, y4), (x3, y3), (255,255,255),1)
    cv2.line(imagen, (x5, y5), (x4, y4), (255,255,255),1)
    cv2.line(Result, (x5, y5), (x4, y4), (255,255,255),1)
    cv2.line(imagen, (xM2, yM2), (x4, y4), (255,0,255),1)
    cv2.line(Result, (xM2, yM2), (x4, y4), (255,0,255),1)

    cv2.line(imagen, (x7, y7), (x5, y5), (255,0,255),1)
    cv2.line(Result, (x7, y7), (x5, y5), (255,0,255),1)
    cv2.line(imagen, (x6, y6), (x5, y5), (255,0,255),1)
    cv2.line(Result, (x6, y6), (x5, y5), (255,0,255),1)
    cv2.line(imagen, (x7, y7), (x6, y6), (255,0,255),1)
    cv2.line(Result, (x7, y7), (x6, y6), (255,0,255),1)
    cv2.line(imagen, (xM3, yM3), (x6, y6), (255,255,255),1)
    cv2.line(Result, (xM3, yM3), (x6, y6), (255,255,255),1)

    # Escribimos en el video los centros:
    cv2.putText(imagen, str(x1), (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(204,204,0),1,cv2.LINE_AA)
    cv2.putText(imagen, str(y1), (x1+50, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(204,204,0),1,cv2.LINE_AA)

    cv2.putText(imagen, str(x2), (x2, y2), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,127),1,cv2.LINE_AA)
    cv2.putText(imagen, str(y2), (x2+50, y2), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,127),1,cv2.LINE_AA)

    cv2.putText(imagen, str(x3), (x3, y3), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,255,0),1,cv2.LINE_AA)
    cv2.putText(imagen, str(y3), (x3+50, y3), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,255,0),1,cv2.LINE_AA)

    cv2.putText(imagen, str(x4), (x4, y4), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,255,0),1,cv2.LINE_AA)
    cv2.putText(imagen, str(y4), (x4+50, y4), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,255,0),1,cv2.LINE_AA)

    cv2.putText(imagen, str(x5), (x5, y5), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,0,255),1,cv2.LINE_AA)
    cv2.putText(imagen, str(y5), (x5+50, y5), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,0,255),1,cv2.LINE_AA)

    cv2.putText(imagen, str(x6), (x6, y6), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,0),1,cv2.LINE_AA)
    cv2.putText(imagen, str(y6), (x6+50, y6), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,0),1,cv2.LINE_AA)

    cv2.putText(imagen, str(x7), (x7, y7), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,128,255),1,cv2.LINE_AA)
    cv2.putText(imagen, str(y7), (x7+50, y7), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,128,255),1,cv2.LINE_AA)

    cv2.putText(imagen, str(xM1), (xM1, yM1), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)
    cv2.putText(imagen, str(yM1), (xM1+50, yM1), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)

    cv2.putText(imagen, str(xM2), (xM2, yM2), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)
    cv2.putText(imagen, str(yM2), (xM2+50, yM2), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)
```

```
    cv2.putText(imagen, str(xM3), (xM3, yM3), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)
    cv2.putText(imagen, str(yM3), (xM3+50, yM3), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)


    cv2.putText(Result, str(x1), (x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(204,204,0),1,cv2.LINE_AA)
    cv2.putText(Result, str(y1), (x1+50, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(204,204,0),1,cv2.LINE_AA)

    cv2.putText(Result, str(x2), (x2, y2), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,127),1,cv2.LINE_AA)
    cv2.putText(Result, str(y2), (x2+50, y2), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,127),1,cv2.LINE_AA)

    cv2.putText(Result, str(x3), (x3, y3), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,255,0),1,cv2.LINE_AA)
    cv2.putText(Result, str(y3), (x3+50, y3), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,255,0),1,cv2.LINE_AA)

    cv2.putText(Result, str(x4), (x4, y4), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,255,0),1,cv2.LINE_AA)
    cv2.putText(Result, str(y4), (x4+50, y4), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,255,0),1,cv2.LINE_AA)

    cv2.putText(Result, str(x5), (x5, y5), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,0,255),1,cv2.LINE_AA)
    cv2.putText(Result, str(y5), (x5+50, y5), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,0,255),1,cv2.LINE_AA)

    cv2.putText(Result, str(x6), (x6, y6), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,0),1,cv2.LINE_AA)
    cv2.putText(Result, str(y6), (x6+50, y6), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,0),1,cv2.LINE_AA)

    cv2.putText(Result, str(x7), (x7, y7), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,128,255),1,cv2.LINE_AA)
    cv2.putText(Result, str(y7), (x7+50, y7), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(0,128,255),1,cv2.LINE_AA)

    cv2.putText(Result, str(xM1), (xM1, yM1), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)
    cv2.putText(Result, str(yM1), (xM1+50, yM1), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)

    cv2.putText(Result, str(xM2), (xM2, yM2), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)
    cv2.putText(Result, str(yM2), (xM2+50, yM2), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)

    cv2.putText(Result, str(xM3), (xM3, yM3), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)
    cv2.putText(Result, str(yM3), (xM3+50, yM3), cv2.FONT_HERSHEY_SIMPLEX, 0.7,
(255,0,255),1,cv2.LINE_AA)



    cv2.putText(Datos, "Angulo 1", (50, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(204,204,0),1,cv2.LINE_AA)
    cv2.putText(Datos, str(gamma1), (230, 50), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(204,204,0),1,cv2.LINE_AA)

    cv2.putText(Datos, "Angulo 2", (50, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(255,0,127),1,cv2.LINE_AA)
    cv2.putText(Datos, str(gamma2), (230, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(255,0,127),1,cv2.LINE_AA)
```

```
    cv2.putText(Datos, "Angulo 3", (50, 110), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0,255,0),1,cv2.LINE_AA)
    cv2.putText(Datos, str(gamma3), (230, 110), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(0,255,0),1,cv2.LINE_AA)




    #Mostramos la imagen original con la marca del centro y
    #la mascara

    cv2.imshow('Camara', imagen)
    cv2.imshow('Result', Result)
    cv2.imshow('Datos', Datos)
    tecla = cv2.waitKey(5) & 0xFF
    if tecla == 27:
        break

cv2.destroyAllWindows()
```

7.7 Anexo VII: Código Python Calibración de Colores[14]

```
import serial
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

def nothing(x):
   pass

#Creamos una ventana llamada 'image' en la que habra todos los sliders
cv2.namedWindow('image')
cv2.createTrackbar('Hue Minimo','image',0,255,nothing)
cv2.createTrackbar('Hue Maximo','image',0,255,nothing)
cv2.createTrackbar('Saturation Minimo','image',0,255,nothing)
cv2.createTrackbar('Saturation Maximo','image',0,255,nothing)
cv2.createTrackbar('Value Minimo','image',0,255,nothing)
cv2.createTrackbar('Value Maximo','image',0,255,nothing)

while(1):
  _,frame = cap.read() #Leer un frame
  hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) #Convertirlo a espacio de color HSV

  #Los valores maximo y minimo de H,S y V se guardan en funcion de la posicion de los
sliders
  hMin = cv2.getTrackbarPos('Hue Minimo','image')
  hMax = cv2.getTrackbarPos('Hue Maximo','image')
  sMin = cv2.getTrackbarPos('Saturation Minimo','image')
  sMax = cv2.getTrackbarPos('Saturation Maximo','image')
  vMin = cv2.getTrackbarPos('Value Minimo','image')
  vMax = cv2.getTrackbarPos('Value Maximo','image')

  #Se crea un array con las posiciones minimas y maximas
  lower=np.array([hMin,sMin,vMin])
  upper=np.array([hMax,sMax,vMax])

  #Crear una mascara con solo los pixeles dentro del rango de verdes
  mask = cv2.inRange(hsv, lower, upper)

  #Mostrar los resultados y salir
  cv2.imshow('camara',frame)
  cv2.imshow('mask',mask)
  k = cv2.waitKey(5) & 0xFF
  if k == 27:
    break
cv2.destroyAllWindows()
```

[14] Código obtenido de: https://robologs.net/2014/07/02/deteccion-de-colores-con-opencv-y-python/

BIBLIOGRAFÍA.

Libros de Robótica:

[1] Barrientos, Antonio, Fundamentos de Robótica, 2ª edición, 2007.

Enlaces de Arobotix – M:

[2] https://learn.trossenrobotics.com/arbotix/7-arbotix-quick-start-guide

[3] http://codigoelectronica.com/blog/como-instalar-arduino-en-raspberry-pi

[4]https://www.ftdichip.com/Support/Documents/AppNotes/AN_220_FTDI_Drivers_Installation_Guide_for_Linux.pdf

Enlaces de RaspBerry Pi y OpenCV:

[5] https://www.pyimagesearch.com/

[6] https://www.pyimagesearch.com/2017/09/04/raspbian-stretch-install-opencv-3-python-on-your-raspberry-pi/

[7] https://robologs.net/2014/07/02/deteccion-de-colores-con-opencv-y-python/

Enlaces de Dynamixel AX-12A:

[8] http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/

[9] http://forums.trossenrobotics.com/showthread.php?7272-Precision-of-AX-12A

Impresión 3D:

[10] http://soloelectronicos.com/2017/10/03/primeros-pasos-con-una-impresora-3d-prusa-i3-pro/

[11] http://imprimalia3d.com/recursosimpresion3d/gu-configuraci-n-par-metros-slic3r

[12] https://www.spainlabs.com/foros/