



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



ESCUELA TÉCNICA
SUPERIOR INGENIEROS
INDUSTRIALES VALENCIA

Curso Académico:

Resumen

En este trabajo se ha desarrollado un brazo robótico de 3 grados de libertad, cuyas piezas han sido fabricadas por impresión 3D y cuyos programas se han creado para el entorno ROS, tanto para PC como para Raspberry Pi, satisfaciendo diversas formas de control cinemático.

En primer lugar, en desarrollo teórico, se realiza una muy breve introducción a los sistemas robóticos. Seguidamente, se exponen los fundamentos matemáticos con que obtener el modelo cinemático, así como distintos tipos de trayectoria. Para comprender mejor las condiciones reales del control cinemático, pasa a explicarse el funcionamiento, estructura y principales parámetros de los servomotores. Por último, se explican los aspectos principales y requeridos de ROS, de Arduino, de Raspberry Pi y del protocolo de comunicación entre dispositivos basado en ROS, *rosserial*.

En segundo lugar, se pasa al desarrollo práctico. Así, se resuelven el problema cinemático directo e inverso. También se explica en este apartado cómo se han adaptado las trayectorias a la realidad limitante de los servomotores. Prosiguiendo con los servomotores, se analiza cada parámetro técnico para elegir los valores más adecuados para el trabajo y se exponen las características de los servomotores elegidos. Ya en el punto de programación, se desarrollan y explican en profundidad los 3 programas creados para ROS con los que controlar el brazo robótico, el software implementado en Arduino, las particularidades de trasladar las operaciones a Raspberry Pi y el funcionamiento de *rosserial*. Para continuar, se trata el diseño del brazo a partir de un modelo previo y de piezas originales; y se enlaza con las ventajas e inconvenientes tanto de la impresión 3D como del material de impresión (PLA). Por último, se describe el ensamblaje del brazo robótico, con algunas eventualidades de interés y se recapitulan las arquitecturas de control conseguidas.

La memoria descriptiva cierra con la conclusiones y reflexiones para proyectos futuros.

Se incluye también un segundo documento con el presupuesto.

Palabras clave: brazo robot, ROS, manipulador, impresión 3D, Raspberry, Arduino, servomotor, paquetes, programación, *rosserial*, fabricación, control cinemático, bajo coste.

Resum

En aquest treball s'ha desenvolupat un braç robòtic de 3 graus de llibertat, les peces del qual han sigut fabricades per impressió 3D i els programes del qual s'han creat per a l'entorn ROS, tant per a PC com per a Raspberry Pi, satisfent diverses formes de control cinemàtic.

En primer lloc, al desenvolupament teòric, es realitza una molt breu introducció als sistemes robòtics. Seguidament, s'exposen els fonaments matemàtics amb què obtindrà el model cinemàtic, així com diferents tipus de trajectòria. Per comprendre millor les condicions del control cinemàtic, passa a explicar-se el funcionament, estructura i principals paràmetres dels servomotors. Per últim, s'hi expliquen els aspectes principals i requerits de ROS, d'Arduino, de Raspberry i del protocol de comunicació entre dispositius basat en ROS, *rosserial*.

En segon lloc, es passa al desenvolupament pràctic. Així, es resolen el problema cinemàtic directe i invers. També s'explica en aquest apartat com s'han adaptat les trajectòries a la realitat limitant dels servomotors. Prosseguint amb els servomotors, s'analitza cada paràmetre tècnic per escollir els valors més adequats pel treball i s'hi exposen les característiques dels servomotors elegits. Ja al punt de programació, es desenvolupen i expliquen en profunditat els 3 programes creats per a ROS amb els quals controlar el braç robòtic, el *software* implementat en Arduino, les particularitats de traslladar les operacions a Raspberry Pi i el funcionament de *rosserial*. Per continuar, es tracta el disseny del braç a partir d'un model previ i de peces originals; i s'enllaça amb els avantatges i inconvenients tant de la impressió 3D com del material d'impressió (PLA). Per últim, es descriu l'assemblatge del braç robòtic, amb algunes eventualitats d'interès i es recapitulen les arquitectures de control aconseguides.

La memòria descriptiva tanca amb les conclusions i reflexions per a projectes futurs.

S'hi inclou també un segon document amb el pressupost.

Paraules clau: braç robot, ROS, manipulador, impressió 3D, Raspberry, Arduino, servomotor, paquets, programació, *rosserial*, fabricació, control cinemàtic, baix cost.

Abstract

In this project we have developed a robot arm with 3 degrees of freedom, the pieces of which have been 3D printed and the programs of which have been created for ROS environment, both for PC and for Raspberry Pi, accomplishing different ways of cinematic control.

First of all, in the theoretical part, we do a really brief introduction about robotic systems. After that, we expose the mathematical bases which are needed to obtain the cinematic model, and different kinds of trajectories. To ensure the better comprehension of the real conditions of this cinematic control, we explain the mechanism, structure and the most relevant parameters of servomotors. Finally, we explain the major and the required issues of ROS, of Arduino and of Raspberry Pi, and of the communication protocol between devices based on ROS, *rosserial*.

Secondly, we move on to the practical part. Thus, we solve the direct cinematic problem and the inverse cinematic problem. We also explain in this part how we have adapted the trajectories to the limitations imposed by the servos. Moving on to the servomotors, we analyze each technical parameter in order to choose the most adequate values for the project and we expose the properties of the chosen servos. Already in the programming part, we develop and deeply explain the 3 programs created for ROS which are used to control the robot arm, the Arduino software, the particular issues of taking the operations to the Raspberry Pi and the functioning of *rosserial*. Hereafter, we explain the arm designing procedure, both from a previous model and with original pieces; and then we link it to the advantages and disadvantages of 3D printing and of the printing material (PLA). Finally, we describe the assembly process of the robot arm, and some interesting issues about it and we recapitulate the control architectures accomplished.

The descriptive memory finishes with the conclusions and thoughts for future projects.

A second document with the economical estimate of the project is also included.

Key words: robot arm, ROS, manipulator, 3D printing, Raspberry, Arduino, servomotor, packages, programming, *rosserial*, manufacturing, cinematic control, low cost.

Diseño, construcción y control de un robot manipulador de 3 grados de libertad de bajo coste para el desarrollo de un manipulador móvil.

ÍNDICE DE LA MEMORIA:

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS	12
1.1- Introducción y motivación	12
1.2-Objetivos	12
CAPÍTULO 2. DESARROLLO TEÓRICO	13
2.1-Los robots.....	13
2.1.1-Qué es un robot	13
2.1.2-Tipos de robots.....	13
2.1.3- Robot desarrollado en el TFG.....	13
2.2- Control y modelo cinemático.....	13
2.2.1- El modelo cinemático	13
2.2.1.1- Sistemas de referencia: rotación, traslación y transformación homogénea	14
2.2.1.2- Aproximaciones al problema cinemático	16
2.2.1.3- Modelización cinemática general de un brazo robot	17
2.2.1.4- Notación de Denavit-Hartenberg.....	18
2.2.2- Control cinemático y trayectorias	20
2.2.2.1- Tipos de trayectoria	21
2.3- Los servomotores	21
2.3.1- ¿Qué es un servomotor?.....	21
2.3.2- Estructura de un servomotor	22
2.3.3- Funcionamiento de un servomotor	22
2.3.3.1- Cómo se le indica el giro al servomotor	22
2.3.3.2- Cómo opera el servomotor para realizar dicho giro	23
2.3.3.2.1- Servomotores analógicos	23
2.3.3.2.2- Servomotores digitales.....	24
2.3.4- Parámetros para la elección de un servomotor	24
2.4- Programación	26
2.4.1- ROS.....	26

2.4.1.1- Nivel de archivos de sistema de ROS.....	27
2.4.1.2- Nivel de gráfico de computación de ROS	27
2.4.2- Arduino	28
2.4.2.1- Manejo de servomotores mediante Arduino.....	28
2.4.3- Comunicación entre Arduino y ROS	29
2.4.3.1- Conexión ordenador-placa en Ubuntu.....	29
2.4.3.2- Rosserial.....	29
2.4.4- Raspberry	30
CAPÍTULO 3. DESARROLLO PRÁCTICO.....	31
3.1- Control cinemático.....	31
3.1.1- Resolución del problema cinemático directo.....	31
3.1.2- Resolución del problema cinemático inverso	33
3.1.2.1- Problema codo arriba-codo abajo	35
3.1.3- Validación del resultado con Matlab	36
3.1.4- Control cinemático y trayectorias	37
3.2- Los servomotores.....	38
3.2.1- Elección de servomotores	38
3.2.2- Análisis para la implementación de los servomotores.....	39
3.3- Programación	39
3.3.1- Desarrollo de programas para Arduino.....	40
3.3.2- Creación de paquetes y nodos de ROS en C++	42
3.3.2.1- Creación del paquete.....	42
3.3.2.2- Desarrollo de los nodos	43
3.3.2.2.1- brazo_def	44
3.3.2.2.2- brazo_def_sinc	51
3.3.2.2.3- trayectoria_lineal_def.....	59
3.3.3- Instalación y uso del protocolo roserial	62
3.3.4- Implementación y uso global de los programas	63
3.3.5- Particularidades del trabajo con Raspberry Pi.....	63
3.4- Rediseño y ensamblaje del brazo robótico	64

3.4.1- Análisis del diseño original	64
3.4.2- Rediseño inicial.....	67
3.4.2.1- Redimensionado de los eslabones 2 y 3	68
3.4.2.2- Articulación correspondiente a q_1	71
3.4.2.3- Error en el refuerzo del eslabón 2.....	73
3.4.3- Impresión 3D de las piezas	73
3.4.4- Ensamblaje, operaciones sobre las piezas y correcciones	74
3.5- Arquitecturas finales	83
3.5.1- PC y Arduino	83
3.5.2- Raspberry Pi y Arduino	83
3.5.3- PC, Raspberry Pi y Arduino	83
3.5.4- Raspberry Pi y módulo controlador de motores	83
CAPÍTULO 4. CONCLUSIONES Y PROYECTOS FUTUROS.....	84
4.1- Conclusiones.....	84
4.2- Proyectos futuros	84
CAPÍTULO 5. BIBLIOGRAFÍA.....	86
5.1- Documentación	86
5.2- Imágenes	88

ÍNDICE DE FIGURAS

Figura 2.1 Rotación de un sistema respecto de otro.	14
Figura 2.2 Transformación homogénea y sistemas de coordenadas.....	17
Figura 2.3 Gráficos de transformación.....	17
Figura 2.4 Ejes no coplanarios.	18
Figura 2.5 Ejes coplanarios.	19
Figura 2.6 Ejes que intersectan.	19
Figura 2.7 Sistema a derechas.....	19
Figuras 2.8 y 2.9 Estructura básica de un servomotor y cable de 3 polos de servomotor.....	22
Figura 2.10 Ejemplo PWM servo analógico	23
Figura 2.11 Ejemplo PWM servo digital	24

Figura 2.12 Esquema de funcionamiento de los mensajes en ROS.....	28
Figura 2.13 Conexión de Arduino a PC con Ubuntu 14.04 con instrucciones de terminal.....	29
Figuras 3.1 y 3.2 Brazo de 3 gdl con información de sistemas de referencia y ángulos de giro	31
Figura 3.3 Esquema de la figura 3.1 recortado.	34
Figura 3.4 Esquema de la figura 3.2 ampliado	34
Figura 3.5 Demostración del problema codo arriba-codo abajo.....	35
Figura 3.6 Código de Matlab del problema cinemático directo.....	36
Figura 3.7 Código de Matlab del problema cinemático inverso.....	36
Figura 3.8 Terminal de Ubuntu correspondiente a <i>rosserial</i>	62
Figura 3.9 Ejemplo de publicación de coordenada. Coordenada $z=0.4$ m. 63	
Figura 3.10 Diseño de partida, de perfil.....	64
Figuras 3.11 y 3.12 Brazo original sin pinza	65
Figuras 3.13 y 3.14 Detalles de servomotores originales	65
Figuras 3.15 y 3.16 Detalles de articulación q_1	66
Figuras 3.17, 3.18 y 3.19 Detalle error de diseño original.....	67
Figura 3.20 Muestra de funcionamiento de OpenSCAD	67
Figura 3.21 Muestra de inconvenientes de pre visualización en OpenSCAD	68
Figura 3.22 Detalle terminación pieza eslabón 2A	69
Figura 3.23 Resultado final del alargamiento de la pieza eslabón 2A.....	70
Figura 3.24 Ejemplo código definitivo para el alargamiento de la pieza eslabón 2A.....	70
Figura 3.24 Redimensionado de la pieza eslabón 3B	71
Figuras 3.25, 3.26 Detalle eslabón 1 original	71
Figuras 3.27, 3.28, 3.29 Detalle eslabón 1 desglosado en piezas	71
Figura 3.30 Eslabón 1C modificado.....	72
Figura 3.31 Eslabón 1A modificado	72
Figura 3.32 Bandejas 1 y 2 recién impresas.....	73

Figura 3.33 Una de las bandejas 3 recién impresa	74
Figuras 3.34 y 3.35 Detalle ajuste con apriete acusado para fijar servomotores	75
Figura 3.36 Detalle viruta de PLA adherida a la broca.....	76
Figuras 3.37 y 3.38 Detalle deformación y corrección del PLA tras taladrado	76
Figura 3.39 Ejemplo de alineación para garantizar coaxialidad tras el taladrado	77
Figura 3.40 Ejemplo montaje articulación con servomotor recibiendo señal eléctrica adecuada	77
Figura 3.41 Fijado del servomotor de q_1	78
Figura 3.42 Preparación de la articulación q_3	78
Figura 3.43 Fijado del servomotor de la articulación q_2	78
Figura 3.44 Diseño del primer intento de pieza para articulación	79
Figura 3.45 Montaje de primera versión de articulación q_1	80
Figura 3.46 Preparación para prueba de la articulación q_3	80
Figura 3.47 Prueba primer diseño para articulación q_2	80
Figura 3.48 Detalle piezas de PLA para articulaciones desgastadas tras pruebas.....	81
Figura 3.49 Detalle piezas servomotor antes y después de limado	81
Figura 3.50 Distintas operaciones para evitar interferencia de tornillos en movimiento articular	82
Figura 3.51 Brazo robótico, montaje final.	82

ÍNDICE DEL PRESUPUESTO:

1-JUSTIFICACIÓN DEL PRESUPUESTO:.....	90
2-ESTUDIO ECONÓMICO:	90
2.1-Costes de Personal:	90
2.2-Material inventariable:.....	91
2.3-Material fungible:.....	92
3- RESUMEN DEL PRESUPUESTO:.....	93

Diseño, construcción y control de un robot manipulador de 3 grados de libertad de bajo coste para el desarrollo de un manipulador móvil.

MEMORIA DESCRIPTIVA

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS:

1.1- Introducción y motivación:

Nuestra sociedad se dota cada día de un entramado tecnológico mayor. Los sistemas automáticos rompieron las barreras que los limitaban a la gran producción fabril, las telecomunicaciones hace mucho que dejaron de limitarse a los proyectos militares y aeroespaciales, para formar una parte de importancia creciente en nuestras vidas.

Los sistemas robóticos no podían quedarse atrás; año tras año los robots van ocupando más parcelas de nuestra actividad cotidiana en todos los ámbitos y van superando las rigideces de construcción y programación, así como el alto coste.

Precisamente en este proceso se enmarca el presente trabajo, con el desarrollo de un robot manipulador versátil pero funcional y de bajo coste, desde herramientas como ROS que tratan de hacer de la programación de robots un asunto universal, común a todos los desarrolladores y productos, y adaptable con facilidad para su uso e integración plena en un manipulador móvil.

1.2-Objetivos:

El objetivo fundamental del presente trabajo es el diseño, fabricación, ensamblaje y control informático de un robot manipulador de 3 grados de libertad, que tenga un coste reducido gracias tanto al sistema de fabricación empleado como a los medios de programación a los que se recurra, y que permita, tanto por sus dimensiones como por su modelo y control cinemáticos, ser implementado sobre la cubierta superior de un robot móvil Summit. Además, es también parte de este objetivo fundamental que tanto a nivel de programación como de configuración, el robot sea lo suficientemente versátil como para sentar una base sólida desde la que desarrollar en mayor profundidad un robot manipulador móvil.

Son objetivos secundarios del trabajo:

-Comprender el papel del modelo cinemático y del control cinemático en los robots manipuladores, y aplicar esos conocimientos a un brazo robótico articulado.

-Estudiar las potencialidades del *software* de código libre y del *hardware* usualmente empleado en robótica, así como adquirir un conocimiento de los mismos tanto a nivel usuario como a nivel desarrollador en ROS para Ubuntu en PC, en ROS para Raspbian en Raspberry Pi, en Arduino y profundizar especialmente en la comunicación por medio de ROS entre PC y Raspberry Pi con Arduino.

-Conocer a fondo a nivel teórico y de control los servomotores, como mecanismo de gran importancia para la implementación de movimiento en los sistemas robóticos.

-Detectar y superar las dificultades y eventualidades del diseño de las piezas necesarias para un robot manipulador, tanto a partir de diseños previo como desde cero.

-Analizar las ventajas e inconvenientes de la impresión 3D como sistema de fabricación.

-Familiarizarse con las complicaciones propias de la fabricación y montaje reales de prototipos, afrontándolas y superándolas con éxito.

CAPÍTULO 2. DESARROLLO TEÓRICO:

2.1-Los robots:

2.1.1-Qué es un robot:

“Un robot es una máquina o ingenio programable, capaz de manipular objetos y realizar operaciones antes reservadas solo a personas” (DRAE, 2014) [1].

2.1.2-Tipos de robots:

Los robots pueden ser clasificados según su ámbito de aplicación en:

-Robots industriales: Según la norma ISO 8373 (2012), un robot industrial se define como aquel *robot controlado automáticamente, reprogramable, con múltiples aplicaciones, manipulador, programable en 3 o más ejes, que puede ser o bien fijado en un sitio o móvil para su utilización en aplicaciones de automatización industrial.* [2]

-Robots de servicio: Según la misma norma, un robot de servicio es aquel que realiza tareas útiles para los humanos o equipamiento, excluyendo las aplicaciones de automatización industrial. [2]

Otra posible clasificación está relacionada con la movilidad del robot:

-Robots manipuladores: Aquellos que tienen fijo uno de sus extremos. Son, por ejemplo, los brazos robot, y se corresponden con la gran mayoría de robots industriales; aunque también hay robots de servicio manipuladores. [3]

-Robots móviles: Son robots que pueden desplazarse por sí mismos. Generalmente se utilizan en aplicaciones tanto industriales como de servicio relacionadas con el transporte o la reproducción de formas de actuar humanas. [3]

2.1.3- Robot desarrollado en el TFG:

En el presente Trabajo Fin de Grado se ha desarrollado un brazo robot de tamaño mediano, de forma que el ámbito de aplicación sería la docencia, por lo que podría encuadrarse como robot de servicios manipulador.

Profundizando más en este tipo de robots, se puede observar que hay de múltiples tipos dependiendo de los tipos de articulación y los sistemas de coordenadas a los que dan lugar sus ejes, pero basta indicar que el robot desarrollado en este trabajo es un brazo articulado, que es aquel que tiene al menos 3 articulaciones que permiten el giro. [4]

El brazo robótico articulado de 3 grados de libertad se podrá implementar sobre un robot móvil Summit.

2.2- Control y modelo cinemático:

2.2.1- El modelo cinemático [5]:

El modelo cinemático directo en el caso de un robot manipulador, como es el brazo robot, se define como el modelo matemático que permite calcular la posición del actuador final a partir de su configuración, esto es, mediante una serie de variables de articulación.

El modelo cinemático inverso en el caso de un robot manipulador, se define como el modelo matemático que permite calcular la configuración con la que alcanzar una determinada posición final.

Es decir, en el modelo directo la incógnita es la posición del elemento terminal del robot y la información conocida es su configuración, mientras que en el modelo inverso la incógnita es la configuración y la información conocida es la posición del elemento terminal.

2.2.1.1- Sistemas de referencia: rotación, traslación y transformación homogénea:

Para resolver el problema cinemático, a cada elemento le corresponderá un sistema de referencia.

Para comprender mejor las herramientas que se utilizarán a continuación, conviene primero detenerse para comprender la modelización matemática de la rotación y la traslación, para lo cual se van a combinar ejes de referencia fijos con ejes de referencia móviles, y se estudiará su relación.

-Rotación [5]:

Se procede a estudiar las matrices de rotación partiendo de un caso concreto:

Sean dos sistemas de referencia: uno fijo (OXYZ) y uno móvil (OUVW), que son inicialmente coincidentes.

En un instante cualquiera, el sistema móvil rota en el sentido contrario a las agujas del reloj alrededor del eje U, de modo que el eje V y el eje W giran un ángulo α respecto a su posición inicial, esto es, respecto al eje Y y al eje Z, respectivamente, como se muestra en la figura 2.1:

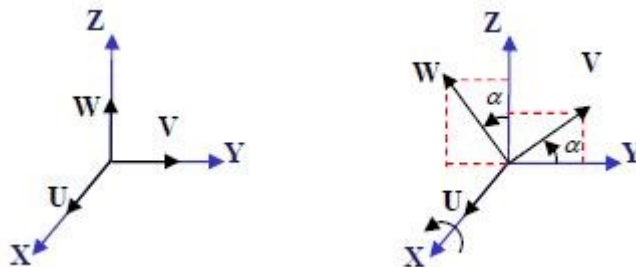


Figura 2.1 [1] Rotación de un sistema respecto de otro.

En el instante posterior a la rotación del sistema móvil, los vectores unitarios de los ejes de ambos sistemas serían:

$$\vec{i}_x = (1,0,0) \quad \vec{j}_y = (0,1,0) \quad \vec{k}_z = (0,0,1)$$

$$\vec{i}_u = (1,0,0) \quad \vec{j}_v = (0, \cos(\alpha), \sin(\alpha)) \quad \vec{k}_w = (0, -\sin(\alpha), \cos(\alpha))$$

Mediante una matriz que recoja en cada elemento la proyección de un eje del sistema de referencia fijo sobre un eje del sistema de referencia móvil (productos escalares de dichos vectores), es posible representar la rotación sufrida por el sistema móvil respecto del fijo. Por este motivo esta matriz recibe el nombre de matriz de Rotación (R):

$$R = \begin{pmatrix} \vec{i}_x \cdot \vec{i}_u & \vec{i}_x \cdot \vec{j}_v & \vec{i}_x \cdot \vec{k}_w \\ \vec{j}_y \cdot \vec{i}_u & \vec{j}_y \cdot \vec{j}_v & \vec{j}_y \cdot \vec{k}_w \\ \vec{k}_z \cdot \vec{i}_u & \vec{k}_z \cdot \vec{j}_v & \vec{k}_z \cdot \vec{k}_w \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}$$

Esto es extrapolable también a un giro alrededor de V y de W.

Generalizando más, estas Matrices de Rotación tienen unas propiedades comunes:

· Cada columna de la matriz se corresponde con un vector unitario de un eje del sistema móvil puesto en función de los vectores unitarios de los ejes del sistema fijo (\vec{i}_u , primera columna, \vec{j}_v , segunda columna, \vec{k}_w , tercera columna).

· Cada fila de la matriz se corresponde con un vector unitario de un eje del sistema fijo puesto en función de los vectores unitarios de los ejes del sistema móvil (\vec{i}_x , primera fila, \vec{j}_y , segunda fila, \vec{k}_z , tercera fila).

· Los vectores fila y columna son unitarios.

· El determinante de la Matriz de Rotación es 1.

· Los productos escalares entre una fila con otra fila, o entre una columna con otra columna, son nulos, debido a que representan vectores ortogonales unos con otros.

· $R^{-1} = R^T$

· Se pueden multiplicar matrices de rotación para representar una secuencia de rotación finita. La multiplicación es no conmutativa:

-Pre multiplica si gira con respecto a OXYZ. ($R=R_{\text{básicaXYZ}} \cdot R$)

-Post multiplica si gira con respecto a OUVW. ($R=R \cdot R_{\text{básicaUVW}}$)

-Traslación [5]:

La traslación de un punto P, ubicado en un sistema coordenado mediante un vector \vec{v} es

$$\vec{P}_{XYZ} = \vec{v} + \vec{P}_{UVW}$$

-Transformación homogénea [5][6]:

Para representar en una matriz tanto rotación como traslación, se introduce la Matriz de Transformación Homogénea T, que representa la orientación y posición del sistema OUVW rotado y trasladado con respecto al sistema OXYZ.

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & \omega_{1x1} \end{bmatrix} = \begin{bmatrix} \text{rotación} & \text{posición} \\ \text{perspectiva} & \text{escalado} \end{bmatrix}$$

En robótica la transferencia de perspectiva es nula, así que el término correspondiente de la matriz, queda anulado, y ω , que representa el escalado global, será 1, de modo que T queda:

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ 0_{1x3} & 1 \end{bmatrix}$$

De forma que se puede representar la posición de un punto P respecto del sistema fijo OXYZ como el producto de la matriz T por las coordenadas de P respecto del sistema móvil OUVW, tan solo introduciendo una coordenada homogénea, ω , que como ya se ha visto, toma el valor 1.

De este modo:

$$\vec{P}_{XYZ} = T \cdot \vec{P}_{UVW} \text{ con } \vec{P}_{XYZ} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \text{ y } \vec{P}_{UVW} = \begin{pmatrix} u \\ v \\ w \\ 1 \end{pmatrix}$$

Con la misma idea en mente, es posible utilizar la matriz T para conocer la posición y orientación final del punto P, $\vec{P}_{XYZ'}$, tras ser rotado y trasladado respecto de su posición inicial referida al sistema de coordenadas fijo OXYZ, \vec{P}_{XYZ} .

De este modo:

$$\vec{P}_{XYZ'} = T \cdot \vec{P}_{XYZ}$$

En este caso, el vector p_{3x1} inserto en la matriz T representará el vector de traslación.

2.2.1.2- Aproximaciones al problema cinemático [5]:

Se distingue entre dos formas de plantear el problema cinemático:

-Aproximación activa: El problema de posición se aborda con desplazamientos de los elementos que componen el robot desde una posición de referencia.

-Aproximación pasiva: El problema de posición se plantea a partir de relaciones entre sistemas de referencia asociados a las barras del robot.

Dado que se opta por esta segunda opción, la aproximación pasiva, se desarrollará en mayor profundidad:

La idea fundamental de este método es que un mismo punto tiene diferentes coordenadas en distintos sistemas de referencia, y que combinando transformaciones se puede pasar la representación del vector de un sistema a otro adyacente, mediante la pre-multiplicación de matrices de transformación homogénea, en la línea de lo visto con anterioridad.

Se define A_n^{n-1} como la matriz de transformación homogénea que nos permite pasar de la representación de las coordenadas de un punto respecto del sistema de referencia n-1 al sistema de referencia n.

Se cumple que: $A_{n-1}^{n-2} \cdot A_n^{n-1} = A_n^{n-2}$

Esta propiedad puede aprovecharse para concatenar distintos sistemas de referencia, ventaja que, como ya puede intuirse, será muy útil en la aproximación pasiva al problema cinemático.

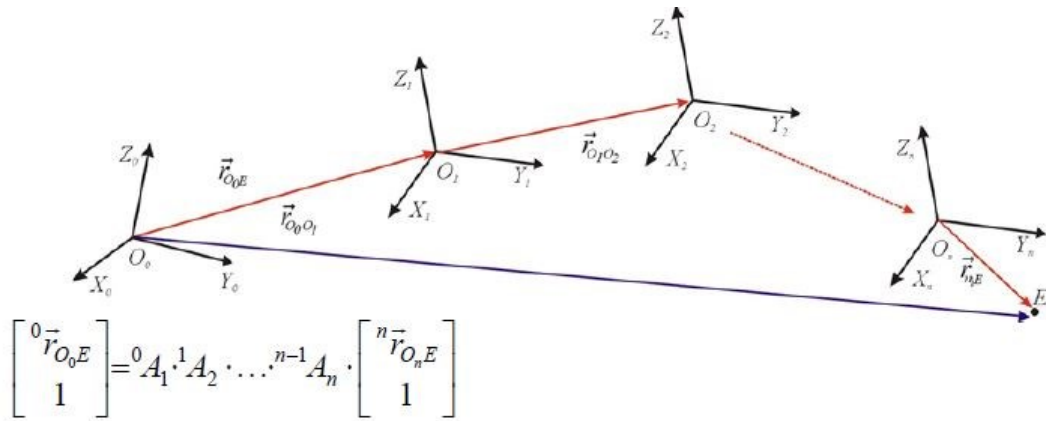


Figura 2.2 [1] Transformación homogénea y sistemas de coordenadas.

2.2.1.3- Modelización cinemática general de un brazo robot [5]:

Un brazo robot constituye, por lo general, una cadena cinemática abierta, esto es, solo existe una secuencia de eslabones conectando los extremos de la cadena; o, dicho de otro modo, el elemento terminal y el inicio del brazo robot se encuentran en puntos distintos .

Para lograr localizar el elemento terminal respecto de un punto que se considere base del brazo robot, se asociará un sistema de referencia móvil a dicho elemento terminal, y se obtendrá la matriz de transformación homogénea:

$$A_n^0 = \begin{pmatrix} \vec{u}_n^0 & \vec{u}_t^0 & \vec{u}_a^0 & \vec{r}_{OP}^0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Donde, como se verá más adelante, nos resultará particularmente útil el vector \vec{r}_{OP}^0 que indica las coordenadas del elemento terminal respecto del sistema de referencia colocado en el punto considerado de partida de la cadena cinemática abierta.

Como hay muchos posibles gráficos de transformación (figura 2.3), es decir, muchos posibles caminos a seguir a la hora de aplicar la transformación homogénea, se recurrirá a un método sistemático, la notación de Denavit-Hartenberg.

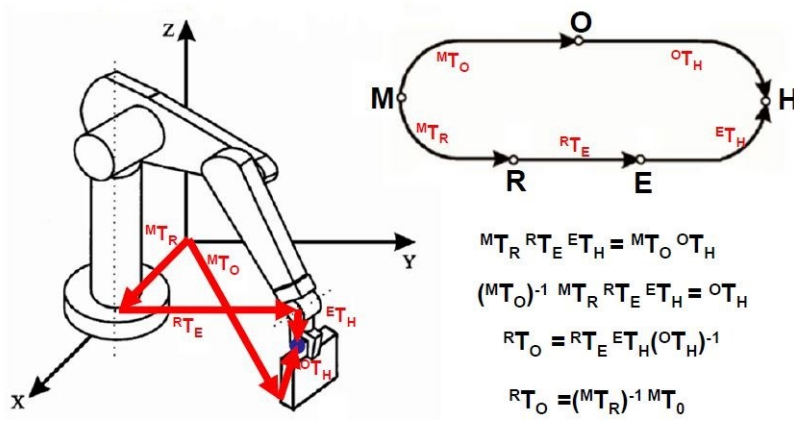


Figura 2.3 [1] Gráficos de transformación.

2.2.1.4- Notación de Denavit-Hartenberg:

La notación de Denavit-Hartenberg es un procedimiento sistemático para colocar sistemas de referencia locales a las barras de un robot de cadena cinemática abierta.

Antes de profundizar en ello, hay que aclarar varias nociones:

-Pares cinemáticos: La Enciclopedia Virtual de Ingeniería Mecánica de la UJI los define como: “*Sistemas de unión entre sólidos rígidos que permiten ciertos movimientos relativos e impiden otros*”. [7]

-Pares cinemáticos de revolución: Aquellos pares cinemáticos que solo permiten el giro de un sólido rígido respecto del otro.

-Pares cinemáticos prismáticos: Aquellos pares cinemáticos que permiten la traslación relativa a lo largo de un eje, pero no el giro.

Una vez aclarados estos conceptos, hay que proseguir con la notación de Denavit-Hartenberg. Los criterios a seguir serán los siguientes, ordenados [5]:

0-Se designará como *barra i* a aquel eslabón que se encuentre entre los nudos i e $i+1$. El sistema de referencia de cada barra se localizará al final de la misma (el sistema i se colocará en el nudo $i+1$, por tanto).

1-El eje Z_i corresponderá al eje característico del par cinemático i .

2-El eje X_i se orienta en función de Z_i y de Z_{i-1} . Hay tres posibles casos:

2.1- Z_i y Z_{i-1} son no coplanarios. En este caso se escoge como eje X_i la normal común de los ejes, con origen en la intersección de la misma con Z_i y normalmente apuntando desde $i-1$ a i .

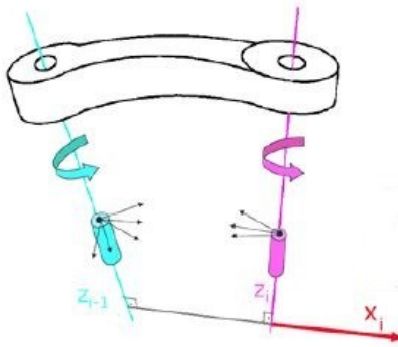


Figura 2.4 [1] Ejes no coplanarios.

2.2- Z_i y Z_{i-1} son paralelos. En este caso se escoge como eje X_i la normal común que pasa por el origen del *frame* $i-1$ y apunta a i , con origen en la intersección con Z_i .

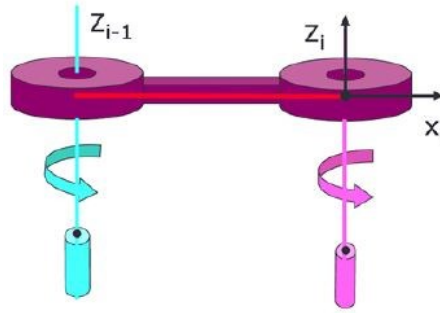


Figura 2.5 [1] Ejes coplanarios.

2.3- Z_i y Z_{i-1} intersectan. En este caso, simplemente se toma X_i como la normal al plano que contiene los ejes.

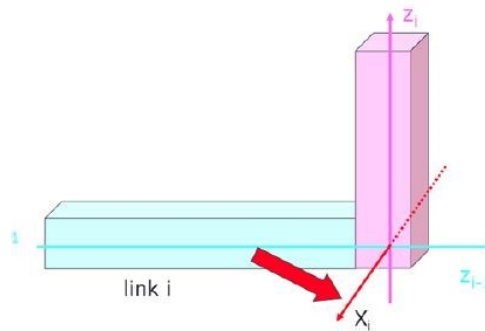


Figura 2.6 [1] Ejes que intersectan.

3-El eje Y_i se coloca en último lugar, de modo que se obtenga un sistema tri-rectangular a derechas (como el de la figura a continuación).

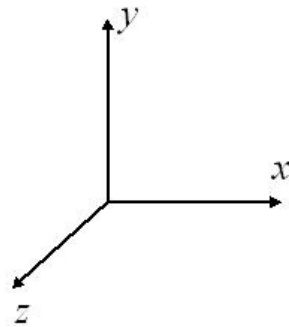


Figura 2.7 [2] Sistema a derechas.

4-Se posicionan los ejes del elemento terminal.

5-Se indican los cuatro parámetros que indican la posición y orientación relativa entre barras adyacentes, solo uno de los cuales puede ser variable (d_i si hay un par cinemático prismático o q_i si hay un par cinemático de revolución):

q_i : Ángulo de la articulación desde x_{i-1} hasta x_i medido desde el eje z_{i-1} usando la regla de la mano derecha.

d_i : Distancia desde O_{i-1} hasta el punto de intersección de z_{i-1} y x_i en la dirección y sentido de z_{i-1} .

a_i : Distancia desde O_{i-1} hasta O_i en la dirección y sentido de x_i .

α_i : Ángulo desde z_{i-1} hasta z_i medido desde el eje x_i .

6-Se tabulan los valores de los parámetros para cada barra, y se obtiene con ellos las matrices de transformación homogénea A_i^{i-1} , que, en su expresión general, toman la forma:

$$\begin{pmatrix} \cos(q_i) & -\cos(\alpha_i) \cdot \sin(q_i) & \sin(\alpha_i) \cdot \sin(q_i) & a_i \cdot \cos(q_i) \\ \sin(q_i) & \cos(\alpha_i) \cdot \cos(q_i) & -\sin(\alpha_i) \cdot \cos(q_i) & a_i \cdot \sin(q_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Esta matriz genérica se obtiene de un sencillo procedimiento de transformación entre dos sistemas de referencia, $\{OXYZ\}_{i-1}$ y $\{OXYZ\}_i$, considerando un sistema intermedio $\{H_i X_i Y_i' Z_{i-1}\}$, donde Y_i' es un eje perpendicular a X_i y a Z_{i-1} .

Para conseguirlo, primero se pasa desde el sistema i hasta el sistema intermedio, con un giro de valor α_i alrededor de x_i (con lo que se logra que Z_i sea paralelo a Z_{i-1}) y con una traslación de valor a_i a lo largo de x_i , de manera que se pasa del sistema i al intermedio.

Este primer paso tiene una matriz de transformación homogénea asociada:

$$A_i^{intermedio} = \begin{pmatrix} 1 & 0 & 0 & \alpha_i \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Y ahora tan solo falta pasar desde el intermedio hasta $i-1$, con un giro de q_i alrededor de $Z_{intermedio}$ (que es paralelo a Z_{i-1}), de modo que $x_{intermedio}$ pasa a ser paralelo con x_{i-1} ; y a continuación se realiza una traslación de valor d_i a lo largo del eje Z' . Con ello, se hace coincidir el sistema intermedio con el $i-1$.

Este segundo paso tiene una matriz de transformación homogénea asociada:

$$A_{intermedio}^{i-1} = \begin{pmatrix} \cos(q_i) & -\sin(q_i) & 0 & 0 \\ \sin(q_i) & \cos(q_i) & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Es fácil ver que, si se aplican secuencialmente ambas operaciones, se está pasando del sistema i al $i-1$, luego, como ya se ha visto anteriormente, se cumplirá:

$$A_i^{i-1} = A_{intermedio}^{i-1} \cdot A_i^{intermedio}$$

2.2.2- Control cinemático y trayectorias:

El control cinemático se encarga de, a partir de una trayectoria de referencia, realizar un muestreo obteniendo puntos finitos pertenecientes a dicha trayectoria y, apoyándose en el modelo cinemático, obtener los valores de las articulaciones en cada uno de esos puntos, de modo que a partir de ellos se obtenga una expresión que se aproxime a esos puntos en función del tiempo y que sea realizable por los actuadores reales que han de implementar dicha trayectoria articular.

2.2.2.1- Tipos de trayectoria [5]:

-Movimiento eje a eje:

Primero se mueve una articulación y cuando termina, se mueve la siguiente, hasta realizar la trayectoria completa. En este caso el tiempo empleado es la suma de lo que tarda cada articulación.

-Movimiento simultáneo de ejes:

Todas las articulaciones comienzan a moverse a la vez, cada una con su velocidad particular, que no tiene por qué ser igual al del resto de articulaciones. En este caso, cada articulación terminará de moverse en un momento diferente, y el tiempo total será el tiempo que tarde la articulación más lenta en alcanzar su configuración final.

-Trayectoria coordinada o isócrona:

Todas las articulaciones comienzan a la vez y terminan a la vez, de modo que adaptan su velocidad para tardar todas un tiempo que, como mínimo, será el de la articulación que tenga una velocidad más limitada.

-Trayectoria continua:

Este caso garantiza que entre el punto de partida y el punto de referencia final, el elemento terminal del robot sigue una trayectoria determinada, por ejemplo, una línea recta, y todas las articulaciones adecuan sus movimientos para asegurar que así sea.

2.3- Los servomotores:

El brazo robot requiere de mecanismos para mover sus articulaciones hasta la posición deseada, con precisión y proporcionando al sistema la fuerza suficiente para moverse y manipular objetos.

Para cumplir estos requisitos, se recurre al uso de servomotores.

Algunas nociones teóricas:

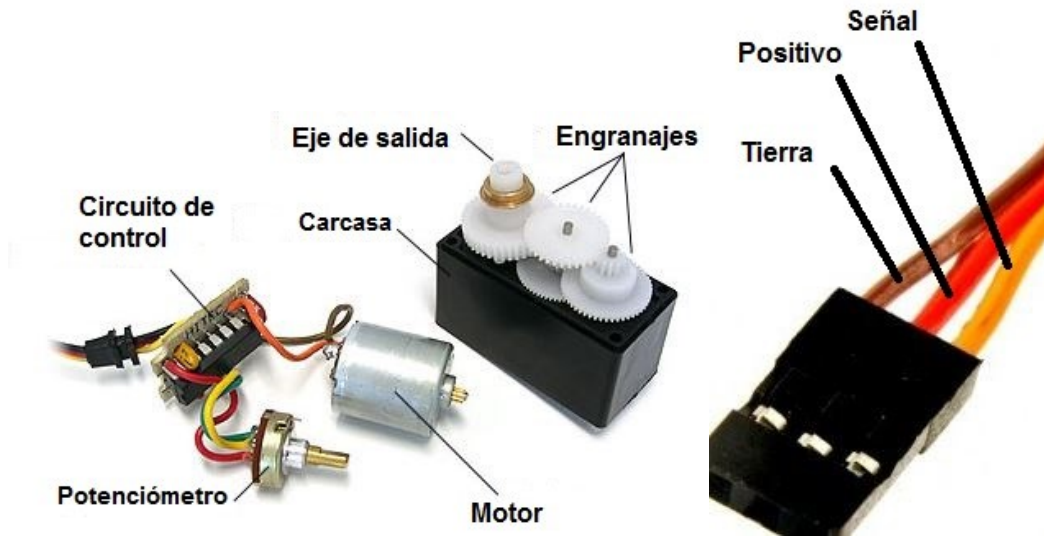
2.3.1- ¿Qué es un servomotor?:

El Laboratorio de Electrónica de la Universidad de Castilla-La Mancha define servomotor como:

“Dispositivo pequeño que tiene un eje de rendimiento controlado. Este puede ser llevado a posiciones angulares específicas al enviar una señal codificada. Con tal de que una señal codificada exista en la línea de entrada, el servo mantendrá la posición angular del engranaje. Cuando la señal codificada cambia, la posición angular de los piñones cambia.” [8]

Es decir, es un dispositivo motor en que puede regularse el ángulo que gira su eje y que mantiene la posición que se le indica.

2.3.2- Estructura de un servomotor [8]:



Figuras 2.8 [3] y 2.9 [4] Estructura básica de un servomotor y cable de 3 polos de servomotor

Como se puede observar en las figuras 2.8 y 2.9, un servomotor está compuesto básicamente por un motor que tiene un potenciómetro conectado a su eje, todo ello gobernado por un circuito de control. El eje del motor se encuentra, a su vez, conectado a un tren de engranajes que reduce la velocidad del motor y la convierte en fuerza en el eje de salida.

Todo ello está contenido en una carcasa.

Al circuito de control llega un cable que, por lo general, tiene 3 polos: con dos se fija la tensión que llegará al motor: tierra y positivo (normalmente, entorno a los 6 V), y con el tercero, se envía la señal con la que se logrará indicar, como se explicará más adelante, al servo el giro que ha de hacer y la fuerza a proporcionar.

La configuración de colores para los cables varía con el fabricante, así que debe consultarse en cada caso.

2.3.3- Funcionamiento de un servomotor:

2.3.3.1- Cómo se le indica el giro al servomotor [8][9]:

La primera parte para entender el funcionamiento de un servomotor, es comprender cómo se le indica la operación que debe realizar:

El método que se utiliza para indicar el giro que debe realizar al servomotor es PWM, *Pulse Width Modulation*.

Esto significa que por el cable de 3 polos, se enviarán pulsos de voltaje positivo que le indicarán la posición que debe alcanzar, de modo que según el ancho de pulso positivo, le corresponderá una posición determinada del eje, y si se cambia dicho ancho de pulso positivo, se accionará el motor para hacer girar el eje hasta la posición correspondiente.

El pulso positivo es el correspondiente al voltaje que llega por el polo de positivo, el valor bajo de la señal será 0, gracias al polo de tierra, y el ancho de pulso vendrá dado gracias a la señal On/Off que llegará por el polo correspondiente a la señal.

Además, el servo cuenta con lo que se denomina como “control proporcional”, es decir, la velocidad de giro aumentará si se ordena un giro mayor. De este modo, el ancho de pulso positivo también controla la velocidad de giro y la fuerza ejercida.

Usualmente un servo puede hacer girar su eje entre 0° y 180°.

Hay que remarcar que en todo momento debe existir una señal con el ancho de pulso correspondiente a la posición que se está ocupando, porque lo que se indica mediante el cable de 3 polos es la posición que debe ocupar, y es el propio servo el que opera para llegar hasta dicha posición (o mantenerse en ella).

2.3.3.2- Cómo opera el servomotor para realizar dicho giro [8]:

El potenciómetro contenido dentro de la carcasa del servomotor está sujeto al eje de salida, y mide en todo momento la posición en la que se encuentra, de modo que sirve de referencia de la posición actual al circuito de control, que será quien le indique cuánto girar al motor.

Hasta aquí, es todo igual para cualquier servomotor, pero en cuanto a cómo procesar la señal regulada por PWM que llega por el cable de 3 polos, hay dos tipos de servomotor diferentes: analógicos y digitales.

2.3.3.2.1- Servomotores analógicos [10]:

En los servomotores analógicos no se proporciona potencia al motor en reposo, pero cuando llega una señal que ordena girar (varía el ancho de pulso positivo) o se intenta forzar de forma externa un cambio de posición respecto del que la señal indica, se envía potencia al motor.

La forma de lograr esto es mediante pulsos positivos de voltaje, de forma que se incrementa el ancho de pulso positivo gradualmente hasta que se logra transmitir todo el voltaje, creando una aceleración del motor que girará el eje del servo hasta su nueva orientación. Cuando se acerca a esta posición, los anchos de pulso positivo empiezan a disminuir para decelerar el motor, hasta pararlo en la orientación deseada.

Estos pulsos siempre se transmiten a una frecuencia de 50 Hz, es decir, 50 ciclos por segundo o, en otras palabras, 50 intervalos de 20 ms cada uno, en los cuales debe haber un pulso positivo que se corresponda con la orientación deseada.

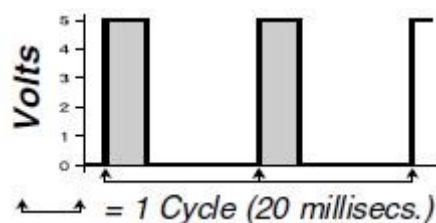


Figura 2.10 [5] Ejemplo PWM servo analógico

Esto lleva al gran problema de los servomotores analógicos: la banda muerta o *Deadband*. Significa que un control de giro pequeño no será efectivo, dado que recurrirá a pulsos

positivos periódicos muy cortos en intervalos de tiempo largos en comparación a ellos, a los que el motor no podrá reaccionar.

Otro problema de los servomotores analógicos es la aceleración o deceleración, puesto que al motor solo le llegan órdenes de aceleración o deceleración cada 20 ms, y esto puede hacer que la aceleración del motor no se ajuste a lo que se desea en cada instante.

2.3.3.2.2- Servomotores digitales [10]:

Los servomotores digitales disponen de un microprocesador que interpreta la información que llega por el cable de 3 polos y la procesa para obtener pulsos de voltaje adecuados para las exigencias del motor y, además, con una frecuencia mucho mayor, de forma que, aunque los pulsos positivos sean más cortos, también serán más cortos los intervalos.

Esto implica que el motor recibirá muchos más pulsos positivos en un segundo, permitiendo una mayor aceleración y reactividad, cosa que además repercutirá positivamente en la forma en que se acelere a cada instante, porque la información/voltaje que llega al motor es actualizada más veces por segundo.

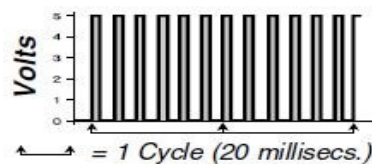


Figura 2.11 [5] Ejemplo PWM servo digital

Los inconvenientes de los servomotores digitales frente a los analógicos es el mayor consumo de potencia y el precio, más alto en el caso de los digitales.

2.3.4- Parámetros para la elección de un servomotor [9]:

-Tamaño:

Micro, en torno a 30x20x30 mm

Standard, en torno a 40x20x40 mm

Gigante, en torno a 60x30x50 mm

Con variaciones dentro de cada tipo.

El tamaño del servo guarda relación con la potencia que puede proporcionar el motor, y esto con la velocidad y fuerza del giro.

-Velocidad para girar 60°:

Da una orientación sobre la rapidez con que el servomotor adquirirá la posición deseada.

Se mide en seg/60°, de forma que cuanto menor sea el valor del parámetro, más rápido será el servo.

Un servo de velocidad normal está alrededor de los 0.24 seg/60°.

-Par:

Máxima fuerza rotacional aplicable a la palanca (brazo del servo).

Se mide en kg·cm, de modo que cuanto más distancia haya entre el centro de rotación (el eje de salida) y el punto en que se hace palanca, menor fuerza se aplicará en dicho punto.

Es uno de los parámetros más importantes a tener en cuenta, y viene condicionado por otros como el tamaño, el voltaje, si es analógico o digital...

-Rango de voltaje de alimentación admisible:

La mayoría admiten entre 4.8V y 6V, aunque los hay que admiten mayores voltajes, siempre entorno a ese orden de magnitud.

A mayor voltaje de alimentación dentro del rango admisible, mayor par proporcionado.

-Analógico o digital

-Material de los engranajes:

Las alternativas suelen ser o materiales metálicos, como el acero, que pesa más pero es más resistente, o materiales plásticos, como el nylon, que son más ligeros pero también menos resistentes.

La resistencia del tren de engranajes condiciona el par que se puede proporcionar, de forma que a mayor resistencia, mayor par proporcionable.

Por tanto, si se busca aumentar el par, habrá que recurrir a engranajes metálicos.

-Peso

Del orden de decenas de gramos, por lo general.

-Tipo de motor:

·Motor común:

Utiliza un núcleo de acero con cables enrollados a su alrededor, e imanes permanentes que lo rodean. Presenta una inercia significativa a la hora de acelerar y decelerar, así como aporta un mayor peso al conjunto del servomotor.

·Motores sin núcleo:

El núcleo de acero del motor común es sustituido por una malla de cables que gira alrededor de los imanes, por su parte exterior. Así, se reduce el peso y, por tanto, la inercia al acelerar y decelerar.

·Motor sin escobillas [11]:

Al eliminar las escobillas que usan el resto de motores eléctricos para invertir la polaridad en el rotor, disminuye el rozamiento y se facilita el mantenimiento, de modo que se gana eficiencia, potencia y vida útil frente a los otros tipos de motor. No obstante, el precio se incrementa, y la inercia es mayor que en los motores sin núcleo.

2.4- Programación:

En el presente trabajo se ha desarrollado el software de control del brazo robot a partir del control sobre los servos, combinando el uso de ROS en Ubuntu-Linux con Arduino.

También se ha implementado el control sobre los servos dominando Arduino desde una Raspberry Pi.

Para entender mejor el trabajo realizado, es necesario explicar algunos de estos conceptos.

2.4.1- ROS [12]:

Como se explica en la enciclopedia virtual de ROS, ROS (Robotics Operating System) es un meta sistema operativo de código abierto para robots. [12]

Es decir, ROS se instala sobre el sistema operativo vigente, principalmente sistemas operativos basados en Unix, y permite que tengan lugar y se comuniquen entre sí distintos procesos, a la vez que se sigue proveyendo todo aquello que cabe esperar de un sistema operativo al uso.

Es de código abierto porque es desarrollado o modificado por los propios usuarios de forma libre, y puesto a disposición del resto de usuarios de forma gratuita.

El principal objetivo de ROS es poder reutilizar código de programación en labores de investigación y desarrollo de robótica. Además, ROS también trata de satisfacer otros objetivos:

-Versatilidad del código escrito para ROS a la hora de utilizarlo en otras herramientas de desarrollo de software para robótica.

-Desarrollar librerías informáticas que ROS pueda utilizar, pero que también sirvan en otras aplicaciones.

-Trabajar con independencia del lenguaje de programación empleado para los distintos programas (actualmente ROS puede trabajar con C++, Python y Lisp).

-Facilitar la puesta a prueba de las aplicaciones.

-Proporcionar una plataforma adecuada para sistemas de gran desarrollo.

Continuando con la información proporcionada por la enciclopedia virtual de ROS, se puede exponer con facilidad los tres niveles conceptuales en que se desarrolla ROS:

-Nivel de archivos de sistema de ROS.

-Nivel de gráfico de computación de Ros: Lo forman los componentes de la red de procesos de ROS que comparten datos.

-Nivel de comunidad: Recursos de ROS que permiten a los usuarios compartir su software y conocimientos. Un ejemplo es la propia enciclopedia electrónica de ROS.

Por la gran cantidad de conceptos que existen en los tres niveles, y dado que el tercero excede las necesidades del trabajo, tan solo se explicarán a continuación aquellos conceptos que son utilizados en el desarrollo del trabajo:

2.4.1.1- Nivel de archivos de sistema de ROS [12]:

-Paquetes (*packages*, en inglés): Unidad básica y mínima de organización de software en ROS. Contiene los *nodos* (procesos individuales), una librería informática dependiente de ROS, archivos de configuración y otros archivos que pueda ser útil organizar juntos.

Para construir un paquete de software utilizable a partir de código fuente, ROS utiliza el sistema de construcción Catkin, que necesita para ello un archivo CMakeLists.txt y un archivo package.xml. Catkin combina Macros de CMake (un sistema de construcción libre y abierto) y *scripts* escritas en Python. [13]

El archivo CMakeLists.txt cuenta con la información que Catkin necesita como entrada para construir los paquetes de software. Incluye la versión mínima de CMake requerida, el nombre del paquete, información para encontrar otros paquetes que puedan ser necesarios para construir el correspondiente, generadores de mensajes, servicios y acciones, generación de invocación de mensajes, servicios y acciones, librerías y archivos ejecutables a construir dentro del paquete, y otras informaciones referentes al paquete. [14]

El archivo package.xml contiene información sobre el paquete, desde su nombre hasta sus autores. Contiene también la declaración de las dependencias del paquete, que son otros paquetes o herramientas de construcción que se necesitan para el paquete. [15]

-Tipos de mensaje: Los mensajes son estructuras de datos sencillas con las que se comunican unos nodos con otros. En un paquete podemos encontrar qué estructuras de datos se usarán por parte de los nodos del paquete.

2.4.1.2- Nivel de gráfico de computación de ROS [12]:

-Nodos (*nodes*, en inglés): Procesos modulares que ejecutan una función y se comunican entre sí intercambiando mensajes (u ofreciendo y usando servicios de ROS). Para controlar un robot, dado el carácter modular de los nodos, normalmente se utilizarán muchos de ellos.

-Maestro (*Master*, en inglés): Permite a los nodos localizarse para comunicarse unos con otros.

-Mensajes (*Messages*, en inglés): Ya explicados.

-Asuntos (*Topics*, en inglés): Buses de intercambio de mensajes. Para que un nodo emita información a otro nodo, el primero deberá ser publicador (*publisher*) y el segundo suscriptor (*subscriber*) del mismo *topic*. Múltiples nodos pueden publicar o suscribirse al mismo *topic* al mismo tiempo.

Para ilustrar mejor la relación entre nodos y *topics*, se añade la figura 2.12, obtenida de la enciclopedia virtual de ROS y modificada para ceñirse al alcance del trabajo:

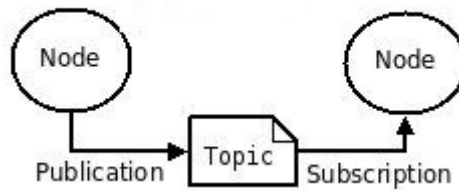


Figura 2.12 [6] Esquema de funcionamiento de los mensajes en ROS

Dado que no se utilizan en el trabajo, no se explicarán otros mecanismos de comunicación entre nodos, como los servicios.

2.4.2- Arduino [16]:

Arduino, como se indica en la página web oficial de Arduino, es una plataforma de implementación de prototipos que se programan en código abierto y que se basa en hardware y software sencillo a la hora de ser utilizado pero versátil para aprovecharlo en distintas aplicaciones.

A nivel de hardware, Arduino cuenta con una amplia colección de placas (boards) con puertos de entrada y salida tanto digitales como analógicos, además de puertos que proporcionan señales de PWM, cosa que las hace particularmente interesantes para el control de servomotores, entre otras funcionalidades. Se les puede conectar gran cantidad de dispositivos, entre ellos, como ya se ha indicado, servomotores.

A nivel de software, Arduino cuenta con su propio lenguaje de programación y con librerías que facilitan ciertas operaciones, tanto matemáticas como de comunicación con dispositivos tales como servomotores, como se tratará más adelante. Para poder programar en Arduino es necesario tener instalado en el ordenador la interfaz gratuita correspondiente.

2.4.2.1- Manejo de servomotores mediante Arduino [16]:

Entre las múltiples librerías incluidas gratuitamente en el software de Arduino, es de particular importancia para el presente trabajo la denominada “servo.h”.

Esta librería permite declarar la conexión de servomotores a los puertos de salida PWM de la placa Arduino y enviar al servomotor la posición que debe adquirir y mantener, o directamente en grados, desde 0° hasta 180°, o bien en ancho de pulso positivo (expresado en microsegundos).

Es importante tener en cuenta que la indicación en grados solo será posible en caso de que la transformación desde microsegundos a grados coincida con la del servomotor que se conecte, cosa que no siempre se da.

Para utilizarla solo es necesario incluirla, declarar la existencia de los servomotores, señalar el puerto al que está conectado cada uno (en la inicialización) y escribir el valor del ángulo deseado en el servo correspondiente.

Hay que alimentar los servomotores desde fuentes de alimentación externas para asegurar que la corriente suministrada no es excesiva para las salidas de voltaje positivo de Arduino. Además, hay que proporcionar una tierra común a Arduino (salida GND), fuente de alimentación y polo de tierra del servomotor.

2.4.3- Comunicación entre Arduino y ROS:

2.4.3.1- Conexión ordenador-placa en Ubuntu [17]:

Para la conectividad de la placa Arduino con ROS es condición necesaria garantizar primero la conectividad de la placa con el ordenador que se está utilizando, y en caso de que dicho ordenador tenga como sistema operativo Ubuntu, como en el presente trabajo (Ubuntu 14.04), ello implica órdenes previas utilizando la Terminal de comandos de Ubuntu.

En la siguiente figura 2.13 se muestra la operación a realizar, que consiste en localizar en qué puerto está conectado el Arduino (primera instrucción), y a continuación conseguir el permiso para utilizar ese puerto USB para utilizarlo con Arduino (segunda instrucción):

```
laboratori@ai2-labrob8:~$ dmesg | grep ttyACM
[ 11.025367] cdc_acm 1-1.3:1.0: ttyACM0: USB ACM device
laboratori@ai2-labrob8:~$ sudo chmod 666 /dev/ttyACM0
[sudo] password for laboratori:
laboratori@ai2-labrob8:~$ █
```

Figura 2.13 Conexión de Arduino a PC con Ubuntu 14.04 con instrucciones de terminal

Al conectarse, pasa a ser posible acceder a la interfaz de Arduino y programar el código que se desee, que deberá ser compilado y descargado a la placa, habiendo indicado previamente a la interfaz qué tipo de placa es y en qué puerto está conectada.

El inconveniente de limitarse a trabajar de esta forma es que una vez descargado el programa, éste funciona de forma automática ejecutando en bucle infinito el código descargado, no permite recibir órdenes desde el ordenador de una forma accesible y estandarizada.

Precisamente la solución a este problema es una de las cuestiones de mayor interés del presente trabajo, la comunicación de Arduino con ROS mediante el protocolo “roserial”.

2.4.3.2- Rosserial:

De acuerdo con la enciclopedia virtual de ROS, roserial es un protocolo informático que trabaja de cara a un dispositivo periférico con los objetivos de [18]:

-Envolver mensajes estándar de ROS con una cobertura informática tal que el periférico pueda procesarlos.

-Funcionar como multiplexor de múltiples *topics* y servicios utilizados para la comunicación con ese dispositivo periférico.

Esto se logra mediante las librerías de cliente (*client libraries*), que permiten que los nodos funcionen con distintos sistemas operativos. Existe una librería general diseñada para comunicarse con microcontroladores, *roserial_client*, que únicamente requiere un compilador ANSI C++ en el ordenador. Además, existen también paquetes que funcionan a modo de extensiones para *roserial_client* específicas para distintos tipos de dispositivo periférico: Arduino, dispositivos que empleen Linux empotrado, dispositivos con aplicaciones en Windows... [19]

En particular, en este trabajo se ha utilizado la extensión *roserial_arduino*, que funciona con Arduino UNO y Arduino Leonardo.

Además, hay una serie de nodos que es necesario activar en el ordenador que sirve de anfitrión para que la comunicación tenga lugar. Estos nodos vienen incluidos en los paquetes: `rosserial_python` (recomendado si el ordenador es un PC), `rosserial_server` (programado en C++) y `rosserial_Java`. En el presente trabajo se utiliza `rosserial_python`, activando el nodo `serial_node.py`. [18]

La forma de lograr la comunicación mediante `rosserial` es utilizando el protocolo a modo de *intermediario*, de manera que, como se tratará a la hora del desarrollo práctico, se vincula el nodo `serial_node.py` al puerto correspondiente al Arduino, y automáticamente `rosserial` envolverá los mensajes que se publiquen en los *topics* a los que esté suscrito o en los que esté publicando el Arduino, para permitir la comunicación con el nodo que utilice también dichos *topics* desde el ordenador.

2.4.4- Raspberry:

Raspberry Pi es, de acuerdo con la información proporcionada por la página web del producto, un ordenador portátil del tamaño de una tarjeta, implementado en una sola placa, con entradas y salidas que permiten conectarla con monitores HDMI, teclado, ratón y otros dispositivos como Arduino, además de *pins* de entrada y salida para conectar otras placas que amplían sus capacidades. [20]

Raspberry Pi puede funcionar con diversos sistemas operativos, pero el más común, recomendado por la propia Fundación Raspberry [20], es Raspbian, un sistema operativo libre y gratuito basado en Debian, inicialmente creado por un grupo de desarrolladores no afiliados a la fundación. Raspbian cuenta con varias versiones, la más moderna de las cuales es Jessie, empleada en este trabajo. [21]

Como Raspbian se basa en Debian, y éste está a su vez basado en GNU/Linux [21], se pueden utilizar muchas herramientas comunes a un ordenador que trabaje con Ubuntu, incluido ROS, lo cual convierte Raspberry Pi en una opción muy atractiva por su portabilidad y versatilidad para el desarrollo de software (y la implementación de hardware) de control de robots.

Además, existe una variante de `rosserial` para utilizar desde Raspberry Pi y otros sistemas monoplaca: `rosserial` para Linux empujado. Esta variante de `rosserial` incluye distintos paquetes, incluyendo `rosserial_arduino`, de forma que se consiguen resultados idénticos a los que se obtendrían con el uso de un ordenador normal. [18]

Cabe destacar que esta variante de `rosserial` no trabaja con las versiones de ROS a partir de Indigo en adelante, de modo que en el presente trabajo se ha utilizado Ros Hydro (la anterior a Indigo, habiendo probado y descartado antes esta).

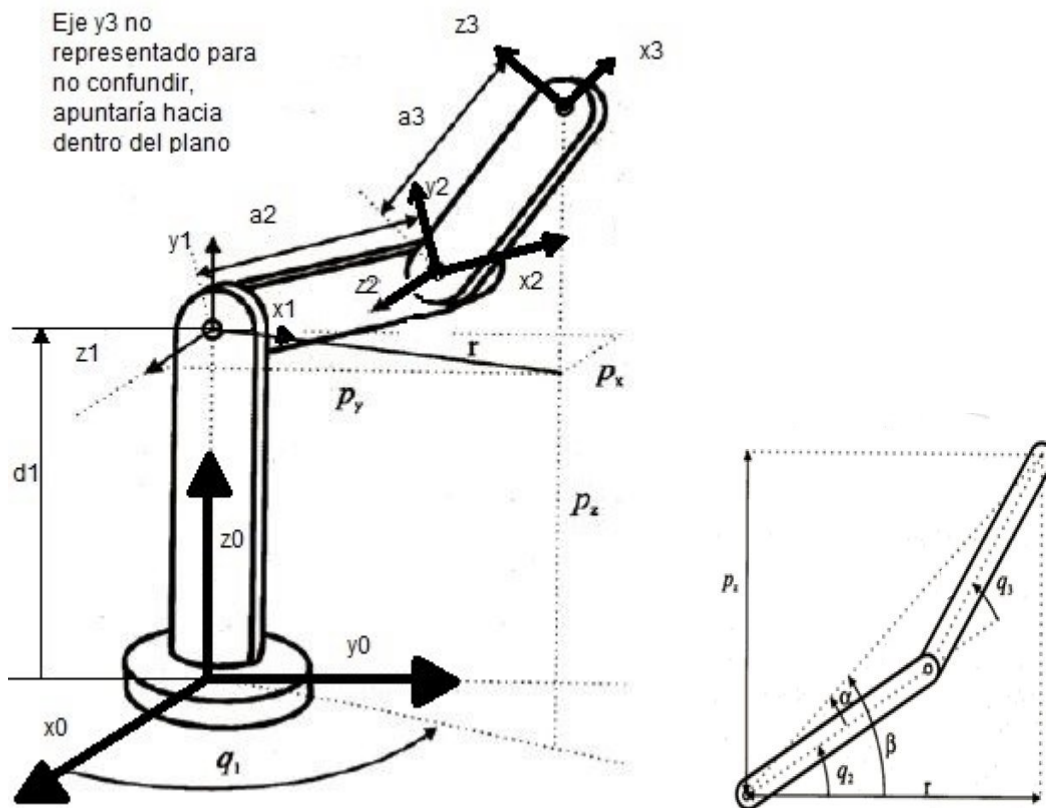
CAPÍTULO 3. DESARROLLO PRÁCTICO:

3.1- Control cinemático:

3.1.1- Resolución del problema cinemático directo:

Para resolver el problema cinemático directo, se recurre a la notación de Denavit-Hartenberg, aplicada a nuestro brazo robot de 3 grados de libertad.

Se adjunta un esquema de colocación de los ejes y parámetros de Denavit-Hartenberg (figuras 3.1 y 3.2) para poder referenciar mejor las explicaciones y el desarrollo. Este esquema no guarda las proporciones en cuanto a las dimensiones reales del brazo, pero sí su configuración cinemática:



Figuras 3.1 y 3.2 [7] Brazo de 3 gdl con información de sistemas de referencia y ángulos de giro

-Colocación de los ejes Z_i :

Se identifican tres pares de revolución, uno con eje de revolución vertical, donde se colocarán Z_0 y otros dos con ejes de revolución paralelos, en los nudos 1 y 2, donde se posicionarán los ejes Z_1 y Z_2 , que lógicamente serán paralelos.

Z_3 se indica de conformidad a lo visto para el elemento terminal, al no haberse de modelizar el comportamiento una garra.

-Colocación de los ejes X_i :

X_0 (e Y_0) se coloca para conseguir un sistema de referencia a derechas que facilite la posterior identificación de parámetros.

X_1 es perpendicular al plano que forman Z_0 y Z_1 , dado que estos ejes intersectan, y se coloca en el punto de intersección (nudo 1).

X_2 es perpendicular tanto a Z_1 como a Z_2 , dado que estos son paralelos, y se coloca siguiendo las indicaciones dadas anteriormente, con sentido i-1 hacia i.

X_3 se coloca con el mismo criterio que X_1 .

-Colocación de los ejes Y_i :

Se posicionan con el fin de que se formen sistemas de referencia a derechas, como se ha indicado en las nociones teóricas.

-Parámetros:

$a_1=0$, dado que no hay separación alguna en la dirección del eje X_0 entre O_0 y O_1 .

a_2 y a_3 se corresponden con las distancias entre nudos 1 y 2, y nudos 2 y 3, respectivamente.

d_1 se corresponde con la altura que hay desde la base hasta el nudo 1, mientras que d_2 y d_3 son nulos, dado que no hay separación entre O_1 y O_2 en la dirección del eje Z_1 , ni entre O_2 y O_3 en la dirección de Z_2 .

$\alpha_1 = -90^\circ$, el ángulo de Z_0 a Z_1 tomando como referencia positiva el eje X_1 .

$\alpha_2 = 0^\circ$, porque $Z_1 \parallel Z_2$.

$\alpha_3 = -90^\circ$, el ángulo de Z_2 a Z_3 tomando como referencia positiva el eje X_3 .

q_1, q_2 y q_3 serán variables, correspondientes a los tres pares cinemáticos de revolución.

-Tabla de Denavit-Hartenberg:

Barra	a_i	α_i	d_i	q_i
1	0	-90°	d_1	q_1
2	a_2	0	0	q_2
3	a_3	-90°	0	q_3

-Matrices de transformación homogénea:

Sustituyendo los valores correspondientes de los parámetros en la matriz de transformación homogénea genérica hallada anteriormente:

$$A_1^0 = \begin{pmatrix} \cos(q_1) & 0 & \text{sen}(q_1) & 0 \\ \sin(q_1) & 0 & -\cos(q_1) & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2^1 = \begin{pmatrix} \cos(q_2) & -\sin(q_2) & 0 & a_2 \cdot \cos(q_2) \\ \sin(q_2) & \cos(q_2) & 0 & a_2 \cdot \sin(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_3^2 = \begin{pmatrix} \cos(q_3) & 0 & -\sin(q_3) & a_3 \cdot \cos(q_3) \\ \sin(q_3) & 0 & \cos(q_3) & a_3 \cdot \sin(q_3) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Se obtendrá así la matriz de transformación homogénea global:

$$T = A_1^0 \cdot A_2^1 \cdot A_3^2$$

Quedando la matriz T (representada como tabla para mayor comodidad):

$[\cos(q_1) \cos(q_2) \cos(q_3)]$ $- [\cos(q_1) \sin(q_2) \sin(q_3)]$	$-\sin(q_1)$	$-[\cos(q_1) \cos(q_2) \sin(q_3)]$ $- [\cos(q_1) \sin(q_2) \cos(q_3)]$	P_x
$[\sin(q_1) \cos(q_2) \cos(q_3)]$ $- [\sin(q_1) \sin(q_2) \sin(q_3)]$	$\cos(q_1)$	$-[\sin(q_1) \cos(q_2) \sin(q_3)]$ $- [\sin(q_1) \sin(q_2) \cos(q_3)]$	P_y
$[\sin(q_2) \cos(q_3)]$ $+ [\cos(q_2) \sin(q_3)]$	0	$-[\sin(q_2) \sin(q_3)]$ $+ [\cos(q_2) \cos(q_3)]$	P_z
0	0	0	1

Donde P_x , P_y y P_z son las coordenadas del elemento terminal, como se indica en el esquema, respecto al sistema 0. La expresión de P_x , P_y y P_z se obtiene también junto al resto de elementos de la matriz T, y es:

$$P_x = a_3 \cos(q_1) \cos(q_2) \cos(q_3) - a_3 \cos(q_1) \sin(q_2) \sin(q_3) + a_2 \cos(q_1) \cos(q_2)$$

$$P_y = a_3 \sin(q_1) \cos(q_2) \cos(q_3) - a_3 \sin(q_1) \sin(q_2) \sin(q_3) + a_2 \sin(q_1) \cos(q_2)$$

$$P_z = a_3 \sin(q_2) \cos(q_3) + a_3 \cos(q_2) \sin(q_3) + a_2 \sin(q_2) + d_1$$

De modo que ya se ha obtenido la solución al problema cinemático directo.

3.1.2- Resolución del problema cinemático inverso:

Dado que la configuración es sencilla de analizar, se obtendrá la solución al problema cinemático inverso simplemente mediante métodos geométricos [5]:

En primer lugar, observando el esquema (figura 3.3), es fácil comprobar que:

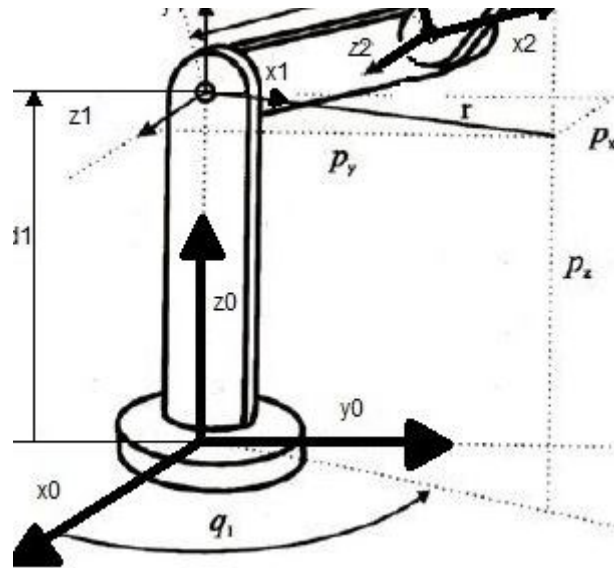


Figura 3.3 [7] Esquema de la figura 3.1 recortado.

$$q_1 = \arctan\left(\frac{p_y}{p_x}\right)$$

A continuación, se obtiene q_3 :

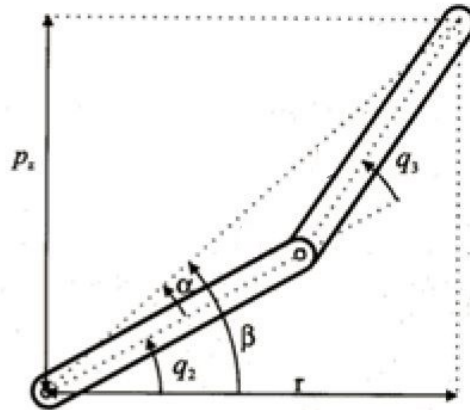


Figura 3.4 [7] Esquema de la figura 3.2 ampliado

En primer lugar, aplicando el teorema de Pitágoras:

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

Y aplicando ahora el teorema del coseno:

$$r^2 + (p_z - d_1)^2 = a_2^2 + a_3^2 + 2a_2a_3 \cos(q_3)$$

Donde a_2 y a_3 serán las mismas longitudes que en el problema cinemático directo.

Despejando esta ecuación, es posible obtener el valor de q_3 :

$$\cos(q_3) = \frac{r^2 + (p_z - d_1)^2 + a_2^2 + a_3^2}{2a_2a_3}$$

$$\sin q_3 = \pm \sqrt{1 - (\cos q_3)^2}$$

$$q_3 = \pm \arctan\left(\frac{\sin q_3}{\cos q_3}\right)$$

Se utiliza la función arctan para resolver adecuadamente el problema del codo arriba o codo abajo, que se explicará a continuación.

Por último, se obtendrá q_2 , para lo cual se recurre a la ayuda del esquema de la imagen anterior:

$$q_2 = \beta - \alpha$$

De modo que tan solo será necesario obtener α y β :

$$\beta = \arctan\left(\frac{p_z - d_1}{r}\right)$$

$$\alpha = \arctan\left(\frac{a_3 \sin(q_3)}{a_2 + a_3 \cos(q_3)}\right)$$

Así, ya se tiene el modelo cinemático, al haber resuelto el problema cinemático directo e inverso.

3.1.2.1- Problema codo arriba-codo abajo:

Un brazo robot articulado puede lograr que su elemento terminal llegue a una misma coordenada con más de una configuración de ángulos en sus articulaciones.

Es común que, en este sentido, en alguna operación trigonométrica surjan dos alternativas de signo opuesto (en este caso, al obtener el ángulo q_3), que dan lugar a dos configuraciones: una que llega a las coordenadas deseadas con el codo (articulación correspondiente a q_2) arriba y otra con el codo abajo, como se muestra en la figura 3.5.

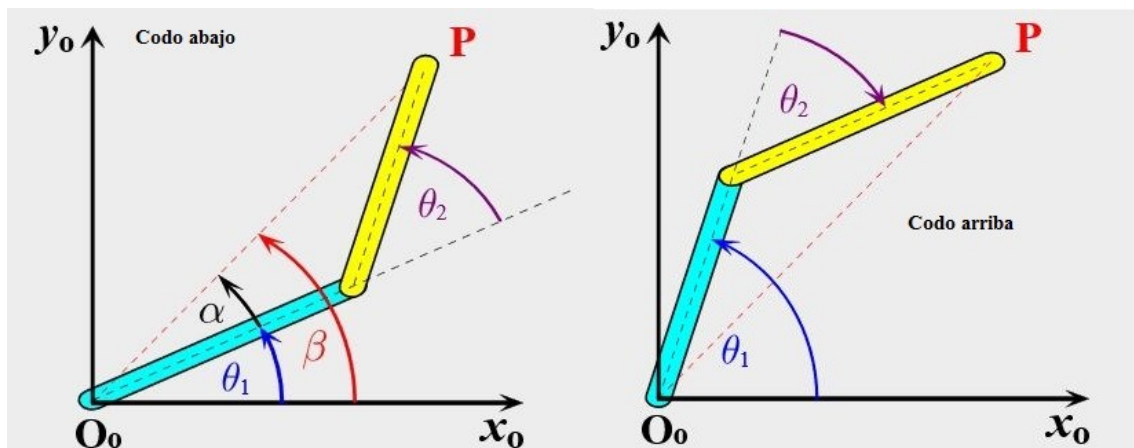


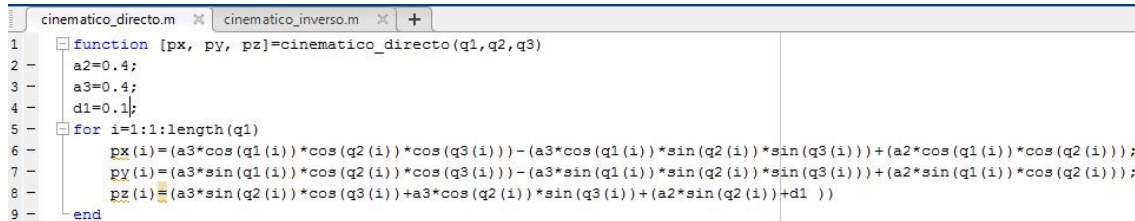
Figura 3.5 [8] Demostración del problema codo arriba-codo abajo

Para solventar este problema, se calcularán ambas alternativas y, una vez obtenido q_2 para ambos casos, se tomará aquel caso que haga que q_2 sea más cercano a 90° . [22]

3.1.3- Validación del resultado con Matlab:

Para asegurarnos de que el resultado obtenido es el correcto, se comprueba su validez con la ayuda de la herramienta informática Matlab.

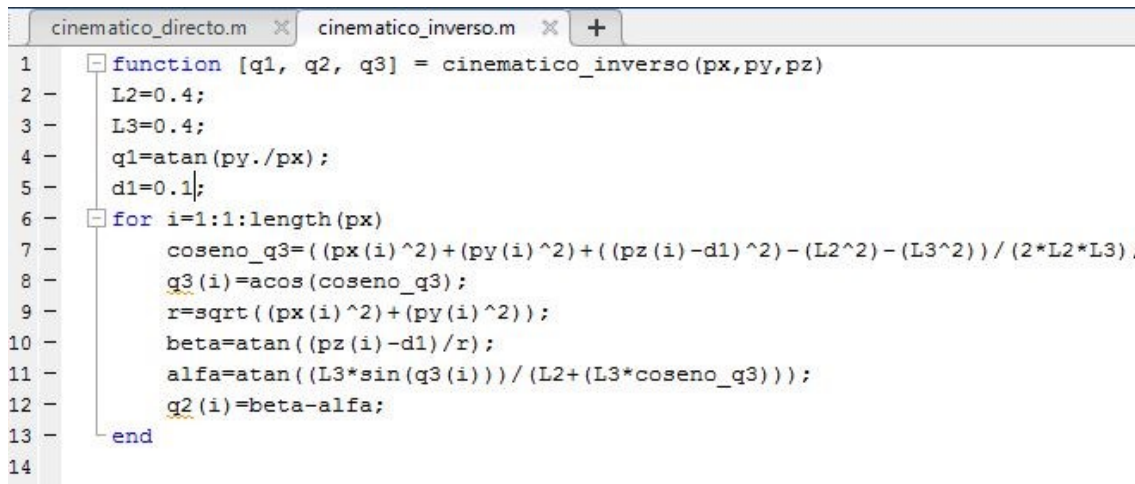
La herramienta de software matemático Matlab ofrece, entre sus muchas funcionalidades, la posibilidad de realizar scripts (programas) escritos con su propio lenguaje de computación y con un asistente. Esta funcionalidad es la que se ha aprovechado para el desarrollo de dos scripts, cuyo código definitivo se reproduce a continuación, que permiten informatizar el modelo cinemático:



```
1 function [px, py, pz]=cinematico_directo(q1,q2,q3)
2 -
3 a2=0.4;
4 a3=0.4;
5 d1=0.1;
6 for i=1:length(q1)
7 - px(i)=(a3*cos(q1(i))*cos(q2(i))*cos(q3(i)))-(a3*cos(q1(i))*sin(q2(i))*sin(q3(i)))+(a2*cos(q1(i))*cos(q2(i)));
8 - py(i)=(a3*sin(q1(i))*cos(q2(i))*cos(q3(i)))-(a3*sin(q1(i))*sin(q2(i))*sin(q3(i)))+(a2*sin(q1(i))*cos(q2(i)));
9 - pz(i)=(a3*sin(q2(i))*cos(q3(i))+a3*cos(q2(i))*sin(q3(i))+(a2*sin(q2(i))+d1));
10 end
```

Figura 3.6 Código de Matlab del problema cinemático directo

En la Figura 3.6 se recoge el código del script que a partir de los ángulos de giro de las articulaciones nos da la posición del elemento terminal. Las ecuaciones implementadas son las obtenidas de la Matriz de Transformación Homogénea a partir de la notación de Denavit-Hartenberg, y, como se observa, gracias al bucle “for”, puede realizar cálculos para una secuencia de valores de los ángulos.



```
1 function [q1, q2, q3] = cinematico_inverso(px,py,pz)
2 -
3 L2=0.4;
4 L3=0.4;
5 q1=atan(py./px);
6 d1=0.1;
7 for i=1:length(px)
8 - coseno_q3=((px(i)^2)+(py(i)^2)+((pz(i)-d1)^2)-(L2^2)-(L3^2))/(2*L2*L3);
9 - q3(i)=acos(coseno_q3);
10 - r=sqrt((px(i)^2)+(py(i)^2));
11 - beta=atan((pz(i)-d1)/r);
12 - alfa=atan((L3*sin(q3(i)))/(L2+(L3*coseno_q3)));
13 - q2(i)=beta-alfa;
14 end
```

Figura 3.7 Código de Matlab del problema cinemático inverso

Desarrollado de una forma muy parecida, el código la Figura 3.7 anterior muestra el cálculo inverso.

Al utilizar los datos obtenidos por una de las funciones como entrada de la otra, se puede comprobar cómo coinciden los valores. Si se introducen unas coordenadas [px,py,pz] en la función de cálculo cinemático inverso, obteniendo unos ángulos (q1,q2,q3), cuando se introduzcan esos ángulos en la función de cálculo cinemático directo, el resultado serán unas coordenadas [px, py, pz] iguales a las iniciales, de forma que se valida el modelo.

Es importante tener en cuenta que en el desarrollo de los programas para su uso práctico habrá que resolver los problemas de indeterminación que pueden darse si existe una división con un 0 en el denominador.

3.1.4- Control cinemático y trayectorias:

En el presente trabajo se han desarrollado matemáticamente y adaptado después a nivel informático tres tipos de trayectoria: movimiento simultáneo de ejes, trayectoria articular isócrona y trayectoria continua rectilínea. Se ha desestimado la trayectoria articular de movimiento eje a eje por entender que no aportaba nada al trabajo.

-Movimiento simultáneo de ejes:

En este caso, como los servomotores trabajan con referencias absolutas de ángulo de referencia, basta con resolver el problema cinemático inverso e indicar a la vez a los tres servomotores qué configuración han de adquirir, de forma que lo harán en el menor tiempo posible, aunque el movimiento es poco recomendable para desplazamientos largos, por la aceleración que adquiere.

-Trayectoria isócrona:

Para implementarla, hay que partir de la base de que los servomotores no tienen una velocidad directamente regulable, sino que su velocidad aumenta con la distancia que han de recorrer.

Para lograr que el movimiento de los servomotores sea sincronizado se toma el ángulo que ha de girar cada uno y con ese ángulo entre la velocidad (0.22 s/60° para el servo Hobbyking y 0.14 s/60° para los servos Goteck) se obtiene un tiempo que será indicativo del tiempo en que cada servomotor realizará la operación. Este tiempo no es exacto, pues la velocidad, como se ha dicho, varía con el ángulo a recorrer, pero permite tomar una referencia del tiempo que, como mínimo, necesitan las articulaciones.

Tomando un tiempo que sea mayor que dicho tiempo mínimo estimado, dividiéndolo entre un periodo de muestreo acorde a las operaciones informáticas a realizar (véase apartado de programación del nodo *brazo_def_sinc*), se logra un número de iteraciones igual para todos los servomotores. Dividiendo el ángulo total a recorrer por el número de iteraciones, se obtiene el incremento angular que deberá darse en cada servo para cada iteración con el objetivo de terminar a la vez.

-Trayectoria continua rectilínea:

Para conseguir esta trayectoria hay que hallar coordenadas (x,y,z) entre las de partida y las de referencia final que se encuentren sobre la misma recta, para que sirvan de micro objetivo a alcanzar.

Estos puntos se calculan con la misma interpolación lineal con la que se extraerían puntos finitos a partir de una trayectoria rectilínea de referencia:

$$p_i = (p_{final} - p_{inicial}) \cdot \frac{i}{\text{número total de puntos}} + p_{inicial}$$

Además, en el presente trabajo se ha asegurado que entre cada micro objetivo de la trayectoria rectilínea, las articulaciones se mueven de forma isócrona.

3.2- Los servomotores:

3.2.1- Elección de servomotores:

En el caso de este servomotor, se necesitan 3 servomotores, uno para cada articulación, donde al menos el correspondiente al giro q_1 y el correspondiente al giro q_2 han de ser capaces de proporcionar un gran par, pues movilizan todo el resto de la configuración.

Además, como se explica en otros apartado de la memoria, un remodelado radical del modelo de partida del brazo robot podría tener implicaciones que excediesen el marco de nuestro trabajo, de modo que habrá que mantenerse en unas dimensiones que respeten aproximadamente la configuración de partida.

Los parámetros determinantes para el brazo robot del presente trabajo serán pues: par mecánico y tamaño.

Se procede a analizar los criterios que se fijan para los distintos parámetros:

-Tamaño:

El modelo de partida cuenta con servomotores estándar de bajo par proporcionado.

-Velocidad para girar 60° :

No es un condicionante en este caso, dado que no se requieren movimientos rápidos.

-Par:

Es el parámetro a maximizar. Un par proporcionado del orden de $25\text{kg}\cdot\text{cm}$ para los servos más potentes y de al menos $10\text{kg}\cdot\text{cm}$ para el otro serían adecuados.

-Rango de voltaje de alimentación admisible:

Interesa mantenerse en los valores estándar, de entre 4.8 a 6V.

-Analógico o digital:

Como ocurre con la velocidad, no es determinante para el desarrollo de esta aplicación.

-Material para engranajes:

Con el objetivo de maximizar el par para un determinado tamaño, habrá que recurrir a engranajes metálicos.

-Peso:

Dentro de lo que cabe, interesa reducirlo, siempre respetando el objetivo del máximo par posible.

-Tipo de motor:

Se supeditará al resto de objetivos.

Proceso de selección:

En primer lugar, se realiza un barrido exhaustivo por la web de dos marcas que comercializan servomotores a nivel mundial: HiTec y Futaba.

Esto se hace con el objetivo de familiarizarse con los valores de parámetro más comunes de cada tipo de servo, con la correlación entre parámetros y con los precios.

A continuación, en páginas de modelismo y robótica se consulta, ya con nociones más claras, para acotar los resultados.

Lo primero que se descartan son los servos Micro, dado que en el mejor de los casos logran un par del orden de 5 kg·cm, muy por debajo de las exigencias, de modo que, por descarte, será necesario recurrir a servos de tamaño Standard.

Entre estos servos también abundan los de bajo par, pero finalmente resulta posible encontrar dos tipos servos con las propiedades deseadas:

Un servomotor “HobbyKing™ High Torque Servo MG/BB W/Proof” de par entre 10 y 12 kg·cm. Este servo será utilizado para la articulación correspondiente al ángulo q_3 .

Dos servomotores “Goteck DC1611S Digital MG High Torque STD Servo” de par entre 20 y 22 kg·cm.

3.2.2- Análisis para la implementación de los servomotores:

Tras realizar el montaje correspondiente con cada uno de los modelos de servomotor, con un sencillo programa de Arduino que recibe un mensaje con la posición indicada en microsegundos y manda esa posición a un puerto PWM, se buscó la correspondencia de grados y microsegundos, resultando ser:

Modelo Goteck: 0° son 900 microsegundos y 120° son 2100 microsegundos.

Así, la transformación viene dada por:

$$\text{microsegundos} = \text{grados} \cdot 10 + 900$$

Modelo Hobbyking: 0° son 650 microsegundos y 180° son 2500 microsegundos.

Y la transformación viene dada por la expresión:

$$\text{microsegundos} = \text{grados} \cdot 10.277 + 650$$

3.3- Programación:

El desarrollo de software del presente Trabajo Fin de Grado abarca cuatro ámbitos fundamentales:

- Desarrollo e implementación de programas de Arduino.
- Creación de paquetes y nodos de ROS en C++ (se ha realizado con ROS *indigo* sobre Ubuntu 14.04).
- Instalación y uso del protocolo roserial.
- Particularidades del trabajo con Raspberry Pi.

Tras exponer los 3 primeros ámbitos, se explicará cómo se trabaja con el conjunto del software y hardware.

3.3.1- Desarrollo de programas para Arduino:

En este apartado se procede a explicar los aspectos más relevantes de la programación en Arduino para el control de los servomotores, así como el programa final. Para agilizar la explicación se hará a partir de una captura del programa final:

```
#if (ARDUINO >= 100)
  #include <Arduino.h>
#else
  #include <WProgram.h>
#endif

#include <Servo.h>
#include <ros.h>
#include <std_msgs/UInt16.h>
#include <std_msgs/Float32.h>

ros::NodeHandle  nh;

Servo servoq1;
Servo servoq2;
Servo servoq3;

void servo_cb( const std_msgs::Float32& cmd_msg){
  servoq1.writeMicroseconds(cmd_msg.data);
}
void servo_cb2( const std_msgs::Float32& cmd_msg){
  servoq2.writeMicroseconds(cmd_msg.data);
}
void servo_cb3( const std_msgs::Float32& cmd_msg){
  servoq3.writeMicroseconds(cmd_msg.data);
}

ros::Subscriber<std_msgs::Float32> sub("servoq1", servo_cb);
ros::Subscriber<std_msgs::Float32> sub2("servoq2", servo_cb2);
ros::Subscriber<std_msgs::Float32> sub3("servoq3", servo_cb3);

void setup(){

  nh.initNode();
  nh.subscribe(sub);
  nh.subscribe(sub2);
  nh.subscribe(sub3);

  servoq1.attach(9);
```

```
servoq2.attach(10);  
servoq3.attach(11);  
}  
  
void loop(){  
  nh.spinOnce();  
  delay(1);  
}
```

Aunque el grueso de la estructura del programa puede ser reproducida a partir de ejemplos que incluye tanto la propia IDE de Arduino como los paquetes de rosserial, hay que detenerse en una serie de aspectos de vital importancia para el trabajo.

En primer lugar, la inclusión de las librerías necesarias para la comunicación mediante mensajes, que son la genérica `ros.h` y las correspondientes a los tipos de mensaje a publicar o a recibir, en este caso, mensajes standard de tipo `int` o `float`, así como la librería para el control de servos [23]:

```
#include <Servo.h>  
#include <ros.h>  
#include <std_msgs/UInt16.h>  
#include <std_msgs/Float32.h>
```

Tras ello, se declaran los servomotores, indicando que el nombre indicado corresponde a un servo [24]:

```
Servo servoq1;
```

A continuación, aparecen varias funciones de nombres derivados de “`servo_cb`”, y las instrucciones de suscripción. Aparecen tres de cada porque hay 3 servomotores a los que enviar datos de posición, pero reproducimos la parte correspondiente a 1 solo servomotor para explicarlo mejor:

```
void servo_cb( const std_msgs::Float32& cmd_msg){  
  servoq1.writeMicroseconds(cmd_msg.data);  
}  
  
ros::Subscriber<std_msgs::Float32> sub("servoq1", servo_cb);
```

En este fragmento de código aparece la definición de una función que recibe un mensaje de tipo `Float32` y envía la posición deseada en forma de ancho de pulso positivo (en microsegundos) al servo denominado “`servoq1`”. También aparece seguidamente la instrucción de ROS que indica que el programa está suscrito al *topic* de ROS llamado “`servoq1`”, recibiendo a través de éste mensajes de ROS tipo estándar que contienen un valor `Float32` y llama a la función “`servo_cb`”, explicada con anterioridad. Así, en estas líneas se entiende cómo Arduino recibe desde el ordenador o la Raspberry Pi un dato de posición y se lo envía a un determinado servomotor [24].

Por último, cabe analizar la siguiente fracción de código, correspondiente a la inicialización del programa:

```
void setup(){  
  
  nh.initNode();  
  nh.subscribe(sub);  
  nh.subscribe(sub2);  
  nh.subscribe(sub3);  
  
  servoq1.attach(9);  
  servoq2.attach(10);  
  servoq3.attach(11);  
}
```

En orden, lo que ocurre es que en las 4 primeras instrucciones se consiguen los *handlers* para que el programa actúe como un nodo de ROS. El primero inicializa como nodo de ROS el programa, y los 3 siguientes consiguen los *handlers* para cada suscripción (nótese que “sub”, “sub2” y “sub3” son nombres fijados en las correspondientes suscripciones implementadas y explicadas anteriormente). Estos handlers son quienes sirven de punto de acceso para las comunicaciones con ROS [24].

En las 3 últimas instrucciones se asocia cada servo a un puerto PWM (en este caso, el 9, 10 y 11).

Como se puede observar en el resto del código, el bucle infinito no hace nada, se limita a ser interrumpido por una de las suscripciones por mensajes que lleguen desde ROS.

3.3.2- Creación de paquetes y nodos de ROS en C++:

3.3.2.1- Creación del paquete:

Partiendo de que se tiene una versión de ROS instalada y un *workspace* o espacio de trabajo de catkin, tal y como se explica en los tutoriales de ros.org [25], se procede a explicar la creación de un paquete de ROS, llamado “brazo_definitivo” y las modificaciones sobre la base del paquete estándar que se han requerido en el presente trabajo.

En primer lugar, se crea el paquete en un *workspace* con las siguientes instrucciones en la terminal de Ubuntu [26]:

```
$ cd ~/catkin_ws/src  
$ catkin_create_pkg brazo_definitivo std_msgs rospy roscpp
```

Así, se genera un paquete llamado *brazo_definitivo* en el directorio *src* del *workspace* *catkin_ws*, con las dependencias *std_msgs* (mensajes estándar), *rospy* y *roscpp*. Este paquete cuenta de partida con un archivo *package.xml* y un archivo *CMakeLists.txt* a los que hay que implementarles las modificaciones indicadas en los tutoriales 3, 10 y 11 sobre ROS de ros.org, correspondientes a la creación de un paquete, el manejo de mensajes y servicios y la creación de un publicador y un suscriptor sencillos [26][27][28][29].

En este caso, cabe concretar que en el CMakeLists.txt se han indicado como ejecutables a construir los 3 nodos contenidos en el paquete. Se expone a modo de ejemplo el código fuente a redactar para uno de ellos al final del CMakeLists.txt del paquete brazo_definitivo:

```
add_executable(brazo_def src/brazo_def.cpp)
target_link_libraries(brazo_def ${catkin_LIBRARIES})
add_dependencies(brazo_def
brazo_definitivo_generate_messages_cpp)
```

En el resto de casos, tan solo hay que sustituir “brazo_def” por el nombre del nodo correspondiente y en la tercera instrucción, “brazo_definitivo_generate_messages_cpp” por la dependencia correspondiente con el nombre del paquete en que se trabaje.

Tras las modificaciones pertinentes en el CMakeLists.txt y el package.xml, y la redacción de los nodos, es necesario ejecutar la orden “catkin_make” en el *workspace* correspondiente, para que *catkin* compile y construya los paquetes, generando los ejecutables indicados. Tras ello, hay que ejecutar la instrucción “source setup.bash” en el directorio *devel* del *workspace*.

```
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash
```

Si no hay errores, será posible ejecutar los distintos programas correspondientes a los nodos mediante la instrucción *roslaunch*, especificando el paquete y el nodo:

```
$ roslaunch brazo_definitivo brazo_def
```

3.3.2.2- Desarrollo de los nodos:

En el presente trabajo se han creado 3 nodos escritos en lenguaje C++ en el paquete “brazo_definitivo”, estos nodos son:

-brazo_def: Simplemente toma por teclado 3 coordenadas (x,y,z) del punto deseado para el elemento terminal del brazo y resuelve el problema cinemático inverso, obteniendo q_1 , q_2 , q_3 y publicándolos en sendos *topics* para que *roslaunch* los adecue de cara a ser implementados con Arduino. Corresponde al modelo de trayectoria articular de movimiento simultáneo de ejes.

-brazo_def_sinc: Además de calcular lo mismo que brazo_def, se asegura de que todos los servomotores llegan a su correspondiente posición final de forma sincronizada en un tiempo que es, como mínimo, el del servomotor más lento, pidiendo por teclado el valor de tiempo deseado y calculando los pequeños incrementos necesarios para que se cumpla ese tiempo de forma exacta, siempre que sea mayor que el del más lento de los servomotores. Corresponde al modelo de trayectoria articular coordinada.

-trayectoria_lineal_def: Similar al anterior pero consiguiendo, además, que el brazo desplace su elemento terminal de un punto inicial a uno final siguiendo dicho elemento una línea recta, y realizando el movimiento en un tiempo indicado por teclado. Corresponde al modelo de trayectoria continua rectilínea.

A continuación se analizarán los 3 programas, explicando su código pero reproduciendo tan solo íntegramente el nodo "brazo_def". De "brazo_def_sinc" se reproducirá el bucle principal y aquellas líneas de programación que difieran de "brazo_def". Por último, de "trayectoria_lineal_def" tan solo se reproducirán los cambios respecto a "brazo_def_sinc", por su similitud en contenido y forma.

3.3.2.2.1- brazo_def:

El código se reproduce a continuación:

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/UInt16.h"
#include "std_msgs/Float32.h"
#include "math.h"

#include <sstream>

float px, py, pz;
bool visto_x, visto_y, visto_z;

void prueba2Callback(const std_msgs::Float32::ConstPtr& msg)
{
    ROS_INFO("px: [%f]", msg->data);
    px=msg->data;
    visto_x=1;
}

void prueba2Callback2(const std_msgs::Float32::ConstPtr& msg)
{
    ROS_INFO("py: [%f]", msg->data);
    py=msg->data;
    visto_y=1;
}

void prueba2Callback3(const std_msgs::Float32::ConstPtr& msg)
{
    ROS_INFO("pz: [%f]", msg->data);
    pz=msg->data;
    visto_z=1;
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "brazo_def");
    ros::NodeHandle n;
    ros::Rate loop_rate(10);
```

```
ros::Publisher prueba2_pub1 =
n.advertise<std_msgs::Float32>("servoq1", 1000);
ros::Publisher prueba2_pub2 =
n.advertise<std_msgs::Float32>("servoq2", 1000);
ros::Publisher prueba2_pub3 =
n.advertise<std_msgs::Float32>("servoq3", 1000);

ros::Subscriber prueba2_sub1 = n.subscribe("teclado1", 1000,
prueba2Callback);
ros::Subscriber prueba2_sub2 = n.subscribe("teclado2", 1000,
prueba2Callback2);
ros::Subscriber prueba2_sub3 = n.subscribe("teclado3", 1000,
prueba2Callback3);

int count = 0;
std_msgs::Float32 q1;
std_msgs::Float32 q2;
std_msgs::Float32 q3;

float var1, var2, var3, varq1, varq2, varq3, px_2, py_2, pz_2,
alfa_1, beta_1, q1g, q2g, q3g, coseno_varq3, seno_varq3_1,
seno_varq3_2, varq3_sol_1, varq3_sol_2, varq2_sol_1,
varq2_sol_2, alfa_2, beta_2, distancia_90_1, distancia_90_2,
q1us, q2us, q3us;
float d1=0.0, a3=0.4, a2=0.4;
while (ros::ok())
{
if(visto_x & visto_y & visto_z){

var1=px;
var2=py;
var3=pz-d1;
if(var1==0){
var1=0.0005;
}
if(var2==0){
var2=0.0005;
}
if(var3==0){
var3=0.0005;
}
px_2=var1*var1;
py_2=var2*var2;
pz_2=var3*var3;
varq1=atan(var2/var1);
coseno_varq3=(px_2+py_2+pz_2-(a2*a2)-(a3*a3))/(2*a2*a3);
seno_varq3_1=sqrt(1-(coseno_varq3*coseno_varq3));
```

```
    if(seno_varq3_1==0){
        seno_varq3_2=seno_varq3_1;
    }
    else{
        seno_varq3_2=-seno_varq3_1;}
    varq3_sol_1=atan(seno_varq3_1/coseno_varq3);
    varq3_sol_2=atan(seno_varq3_2/coseno_varq3);

    alfa_1=atan((a3*sin(varq3_sol_1))/(a2+(a3*cos(varq3_sol_1))
));
    beta_1=atan(var3/sqrt(px_2+py_2));
    varq2_sol_1=beta_1-alfa_1;

    alfa_2=atan((a3*sin(varq3_sol_2))/(a2+(a3*cos(varq3_sol_2))
));
    beta_2=atan(var3/sqrt(px_2+py_2));
    varq2_sol_2=beta_2-alfa_2;
    distancia_90_1=(90-varq2_sol_1);
    distancia_90_2=(90-varq2_sol_2);
    if(distancia_90_1<distancia_90_2){
        varq3=varq3_sol_1;
        varq2=varq2_sol_1;
    }
    if(distancia_90_1>=distancia_90_2){
        varq3=varq3_sol_2;
        varq2=varq2_sol_2;
    }
    q1g=varq1*(180/3.1416);
    q2g=varq2*(180/3.1416);
    q3g=varq3*(180/3.1416);
    q3g=q3g+90;
    q1us= (10*q1g)+900;
        if(q1us>2100){
            ROS_INFO("q1 saturando, introduzca otras
coordenadas");
            q1us=2100;
        }
        else if(q1us<900){
            ROS_INFO("q1 saturando, introduzca otras
coordenadas");
            q1us=900;
        }
    q2us=(10*q2g)+900;
        if(q2us>2100){
            ROS_INFO("q2 saturando, introduzca otras
coordenadas");
            q2us=2100;
        }
        else if(q2us<900){
```

```
        ROS_INFO("q2 saturando, introduzca otras
coordenadas");
        q2us=900;
    }
    q3us=(10.277*q3g)+650; //HK 0 a 180° son 650 a 2500
microsegundos
    if(q3us>2500){
        ROS_INFO("q3 saturando, introduzca otras
coordenadas");
        q3us=2500;
    }
    else if(q2us<650){
        ROS_INFO("q3 saturando, introduzca otras
coordenadas");
        q2us=650;
    }

    q1.data=q1us;
    q2.data=q2us;
    q3.data=q3us;
    visto_x=0;
    visto_y=0;
    visto_z=0;
}
prueba2_pub1.publish(q1);
prueba2_pub2.publish(q2);
prueba2_pub3.publish(q3);

ros::spinOnce();
loop_rate.sleep();
++count;
}

return 0;
}
```

En cuanto a la explicación del código, se comienza por incluir las librerías genéricas de ROS, las de los mensajes a utilizar y las necesarias para el cálculo matemático:

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include "std_msgs/UInt16.h"
#include "std_msgs/Float32.h"
#include "math.h"
```



```
#include <sstream>
```

Seguidamente, se declaran como variables globales las coordenadas y las funciones booleanas asociadas a ellas. Es necesario que sean globales porque van a utilizarse tanto desde el bucle principal del programa como desde las funciones de la suscripción:

```
float px, py, pz;  
bool visto_x, visto_y, visto_z;
```

A continuación, hay 3 definiciones de función que siguen el mismo modelo que las funciones de suscripción del programa de Arduino, ya explicadas. Reciben un mensaje de tipo Standard con un dato Float32 y lo guardan en la variable de la coordenada correspondiente [29], activando una función booleana cuyo objetivo será explicado más adelante. También informan por la terminal del valor de la variable. Se reproduce tan solo una a modo de ejemplo:

```
void prueba2Callback(const std_msgs::Float32::ConstPtr& msg)  
{  
  ROS_INFO("px: [%f]", msg->data);  
  px=msg->data;  
  visto_x=1;  
}
```

El código que sigue corresponde a la función *main* del programa. Nada más empezar, se inicializa el nodo indicando su nombre y consiguiendo un *handler* (como en el programa de Arduino) [29]:

```
ros::init(argc, argv, "brazo_def");  
ros::NodeHandle n;
```

Vale la pena detenerse en la siguiente línea de código:

```
ros::Rate loop_rate(10);
```

En ella se indica la frecuencia con que se ha de reproducir el bucle principal (10 Hz en este caso) [29]. En programas ligeros como este, esta frecuencia puede ser muy alta. Se volverá a tratar esta instrucción en el siguiente nodo, donde toma una importancia particular.

El programa continúa con la indicación al *ros master* de las publicaciones y suscripciones del programa. En la publicación hay que indicar el tipo de mensaje a publicar, mientras que en la suscripción se indica la función a la que se llama para gestionar el mensaje entrante por la suscripción. En ambos casos se asocia un nombre a cada publicación o suscripción, un *topic* y una longitud de cola de publicación o suscripción (1000 mensajes por *topic* en este caso) [29]. Se reproduce una de cada a modo de ilustración:

```
ros::Publisher prueba2_pub1 =  
n.advertise<std_msgs::Float32>("servoq1", 1000);
```

```
ros::Subscriber prueba2_sub1 = n.subscribe("teclado1", 1000,
prueba2Callback);
```

Cada vez que se reciba un mensaje en uno de estos *topics*, el *ros master* realizará el traspaso de información que se le haya demandado desde el nodo.

Antes de entrar en el bucle, se declaran las variables y mensajes que se van a utilizar a nivel local en la función (no se reproduce de nuevo para no sobrecargar la Memoria).

En cuanto al bucle principal, se puede observar que la condición de continuidad del bucle es “while(ros::ok())”. Esta condición solo dejará de cumplirse cuando la función *ros::ok()* devuelva un valor 0, cosa que ocurre únicamente en caso de que se interrumpa desde la terminal el programa (ctrl+c), se inicie un nodo con el mismo nombre que el que está ejecutándose, se llame a la función *ros::shutdown()* desde otro punto de la aplicación o se destruyan todos los *handlers*. Es decir, hasta que el programa se interrumpa voluntariamente, el bucle será ejecutado indefinidamente [29].

En cuanto al contenido del bucle, gracias a la primera condición “if”, el programa calculará una nueva posición del brazo robótico solo si todas las coordenadas han sido actualizadas. Una vez cumplida esta condición, almacena los valores de las variables globales *px*, *py* y *pz*, y resuelven con ellos el problema cinemático inverso. Es de remarcar que, para evitar problemas de indeterminación matemática en caso de coordenadas que sean 0 y puedan dar lugar a divisiones con 0 en el denominador, se sustituyen las coordenadas 0 por un valor cercano a este y tan pequeño que no genera un error significativo:

```
if(visto_x & visto_y & visto_z){
    var1=px;
    [...]
    if(var1==0){
        var1=0.0005;
    }
}
```

En cuanto a la resolución del problema cinemático inverso, reproduce exactamente el modelo teórico ya tratado en el apartado correspondiente, resolviendo además mediante una comparación “if” el problema codo arriba-codo abajo para escoger siempre la opción de codo arriba, es decir, aquella en que el ángulo q_2 se acerque más a los 90° :

```
distancia_90_1=(90-varq2_sol_1);
distancia_90_2=(90-varq2_sol_2);
if(distancia_90_1<distancia_90_2){
    varq3=varq3_sol_1;
    varq2=varq2_sol_1;
}
if(distancia_90_1>=distancia_90_2){
    varq3=varq3_sol_2;
    varq2=varq2_sol_2;
}
```

```
}
```

Tras ello, se convierten los valores obtenidos en radianes a grados, y de grados a microsegundos. La conversión se ha realizado con este paso intermedio porque la recta de transformación en caso de pasar de radianes a microsegundos es mucho menos exacta, dado que utiliza más veces el número π , generando más error de cálculo.

Además, el programa satura en caso de que se le exija al servomotor un ángulo que no puede conseguir por sus limitaciones mecánicas. Se reproduce el ejemplo para uno de los servomotores:

```
q1g=varq1*(180/3.1416);  
[...]  
q1us= (10*q1g)+900;  
    if(q1us>2100){  
        ROS_INFO("q1 saturando, introduzca otras  
coordenadas");  
        q1us=2100;  
    }  
    else if(q1us<900){  
        ROS_INFO("q1 saturando, introduzca otras  
coordenadas");  
        q1us=900;  
    }  
}
```

En el caso del servo q3, como el modelo matemático trabaja con grados entre -90° y 90° , pero tan solo se pueden enviar grados entre 0° y 180° al servomotor, se realiza una sencilla conversión:

```
q3g=varq3*(180/3.1416);  
q3g=q3g+90;
```

Las instrucciones contenidas en el “if(visto_x & visto_y & visto_z)” terminan con el almacenamiento de los valores calculados en microsegundos en los objetos de mensaje estándar que se publicarán, y el reinicio de las booleanas de “visto”, para que no se repitan todas las operaciones hasta que no se introduzcan nuevas coordenadas.

De esto solo se requiere hacer hincapié en que, si bien en la función que gestiona las suscripciones el modo de acceder a los datos del mensaje recibido es “msg->data”, en el caso de estos mensajes que se han definido en la función *main* se accede a sus datos siguiendo el esquema “nombre_del_mensaje.data”. Por ejemplo:

```
q1.data=q1us;
```

Para terminar, se publican los mensajes utilizando la instrucción “.publish” asociada al nombre que se había indicado en la inicialización en el *main* [29]:

```
prueba2_pub1.publish(q1);
```

El motivo por el cual la publicación se da fuera de toda condición, solo al final del bucle principal, es que el programa no funcionaría bien si se reprodujera esta instrucción más de una vez por bucle, incluso dentro de condiciones mutuamente excluyentes.

Ello implica que no puede introducirse la instrucción de publicación dentro de otro bucle “while” o “for”, y que tampoco puede repetirse en dos “if”. Esto es de vital importancia, porque, como se verá más adelante, condiciona toda la estructura de aquellos programas que, por ejemplo, publican referencias que pasan gradualmente de un valor a otro.

En el caso de este programa, no se da esta situación, pero se ha dejado fuera de todo condicionante para que sea más sencillo desarrollar nuevos programas a partir de su código, como de hecho ha sido necesario para los siguientes nodos.

Es importante añadir al final las instrucciones “spin once”, que habilita las funciones de gestión de suscripción mientras el programa está en el bucle infinito, y “sleep”, para cumplir con la frecuencia del bucle que se la ha indicado anteriormente:

```
ros::spinOnce();
loop_rate.sleep();
```

3.3.2.2- brazo_def_sinc:

Tan solo se reproducen las diferencias con el anterior nodo y la mayor parte del bucle principal.

```
#include "ros/ros.h"
[...]
float px, py, pz, tiempo;
bool visto_x, visto_y, visto_z, visto_t;
bool primeravez=1, t_adq=0;
int i=0;
[...]
void prueba2Callback4(const std_msgs::Float32::ConstPtr& msg)
{
  ROS_INFO("tiempo exigido: [%f]", msg->data);
  tiempo=msg->data;
  visto_t=1;
}

int main(int argc, char **argv)
{
  ros::init(argc, argv, "brazo_def_sinc");
  ros::NodeHandle n;
  ros::Rate loop_rate(100);
  [...]
  float [...] t1, t2, t3, tmax, n_it, incre1, incre2, incre3,
  refq1, refq2, refq3, q1g0, q2g0, q3g0, vart;
  float d1=0.0, a3=0.4, a2=0.4;
  float tm=0.01;

  while (ros::ok())
```

```
{
if(primeravez==1){
    printf("Inicializando\n");
    q1g=0;
    q2g=0;
    q3g=90; //0+90
    primeravez=0;
    refq1=q1g;
    refq2=q2g;
    refq3=q3g;
    q1g0=q1g;
    q2g0=q2g;
    q3g0=q3g;
}

if(visto_x & visto_y & visto_z & visto_t){

    vart=tiempo;
    var1=px;
    if(var1==0){
        var1=0.0005;
    }
    var2=py;
    if(var2==0){
        var2=0.0005;
    }
    var3=pz-d1;
    if(var3==0){
        var3=0.0005;
    }
}

[...]
if(q1g>q1g0){
t1=(q1g-q1g0)*0.14/60;}
else{
t1=(q1g0-q1g)*0.14/60;}
tmax=t1;
if(q2g>q2g0){
t2=(q2g-q2g0)*0.14/60;}
else{
t2=(q2g0-q2g)*0.14/60;}
if(t2>tmax){
    tmax=t2;}
if(q3g>q3g0){
t3=(q3g-q3g0)*0.22/60;}
else{
t3=(q3g0-q3g)*0.22/60;}
if(t3>tmax){
    tmax=t3;}
if(tmax<vart && tmax>0){
```

```
        tmax=var_t;
    }
    n_it=tmax/tm;

    if(n_it>=1){
        incre1=(q1g-q1g0)/n_it;
        incre2=(q2g-q2g0)/n_it;
        incre3=(q3g-q3g0)/n_it;

        if(i<=n_it){

            refq1=(incre1*i)+q1g0;

            refq2=(incre2*i)+q2g0;

            refq3=(incre3*i)+q3g0;

            i++;
        }
        if(i>n_it){
            i=0;
            visto_x=0;
            visto_y=0;
            visto_z=0;
            visto_t=0;
            q1g0=q1g;
            q2g0=q2g;
            q3g0=q3g;
        }
    }
    else if(n_it==0){
        i=0;
        visto_x=0;
        visto_y=0;
        visto_z=0;
        visto_t=0;
    }
}

    q1us= (10*refq1)+900; //Goteck 0 a 120° son 900 a 2100
microsegundos
    if(q1us>2100){
        ROS_INFO("q1 Saturando, introduzca otras
coordenadas");
        q1us=2100;
    }
    else if(q1us<900){
        ROS_INFO("q1 Saturando, introduzca otras
coordenadas");
```

```
        q1us=900;
    }
    q2us=(10*refq2)+900;
    if(q2us>2100){
        ROS_INFO("q2 saturando, introduzca otras
coordenadas");
        q2us=2100;
    }
    else if(q2us<900){
        ROS_INFO("q2 saturando, introduzca otras
coordenadas");
        q2us=900;
    }
    q3us=(10.277*refq3)+650; //HK 0 a 180° son 650 a 2500
microsegundos
    if(q3us>2500){
        ROS_INFO("q3 saturando, introduzca otras
coordenadas");
        q3us=2500;
    }
    else if(q2us<650){
        ROS_INFO("q3 saturando, introduzca otras
coordenadas");
        q2us=650;
    }
    }

    q1.data=q1us;
    prueba2_pub1.publish(q1);
    q2.data=q2us;
    prueba2_pub2.publish(q2);
    q3.data=q3us;
    prueba2_pub3.publish(q3);

    ros::spinOnce();
    loop_rate.sleep();
    ++count;
}

return 0;
}
```

Se procede a analizar los aspectos más importantes del código:

En primer lugar, cabe señalar cómo se ha añadido una nueva variable global, “tiempo” con una nueva booleana asociada y una nueva suscripción desde la que se recibe un mensaje de tipo standard Float32 que se gestiona con otra función de “CallBack”, cargando este valor en la variable global “tiempo”. Todo ello sigue la estructura del anterior nodo, así que no se explicará en mayor profundidad.

Este tiempo es aquel en que los servomotores habrán de llegar sincrónicamente a su posición final.

En la función *main* puede observarse que cambia el nombre del nodo respecto al anterior, y, lo que es más importante, que se ha modificado la frecuencia del bucle. Además, más adelante se ha declarado una variable tipo Float, “tm”, el tiempo de muestreo, que hay que explicar conjuntamente:

```
ros::Rate loop_rate(100);  
    [...]  
float tm=0.01;
```

Como se observa, la frecuencia de 100Hz implica que el bucle realizará todas sus operaciones en 0.01 segundos (10 ms). Por ello, cuando quiera realizarse una serie de operaciones mediante diversas iteraciones en un tiempo fijo, habrá de garantizarse que la frecuencia en que el programa realiza las operaciones se corresponde con el tiempo en que se da por supuesto que van a realizarse.

Dicho en otras palabras, el tiempo de muestreo ha de coincidir con el tiempo que tarda el bucle en ejecutar todas sus operaciones si se quiere una adecuación exacta al tiempo indicado.

Para hallar el mínimo tiempo en que tanto ROS como los servomotores pueden operar, se realiza una comprobación, empezando por el rango de operación de ROS entre valores de frecuencia del bucle que resultan de interés, entre 10 y 200 Hz.

Frecuencia del bucle (Hz)	Tiempo de muestreo (microseg)
10	100
20	50
50	20
100	10
200	5

Al indicar como tiempo objetivo para realizar todas las operaciones del bucle principal 4 segundos, se observa que para todos ellos hay un cierto retraso de cerca de 1 segundo.

Para comprobar si es debido a que todo el programa es lento, o tan solo se debe a una suma de error de medida y un breve retraso al inicio de los cálculos, se indica como tiempo a conseguir 40 segundos, de modo que, en caso de que el programa sea lento, no se retrasará solo 1 segundo, sino que el retraso crecerá proporcionalmente.

Tras la prueba, en todos los casos el resultado es correcto, no llegando a retrasarse ni tan siquiera 1 segundo, incluso contando con el error de cronometrado. Es decir, ROS puede realizar las operaciones en todo el rango de frecuencias sin ningún problema.

En cuanto a los servomotores, se descartan los valores extremos, buscando rapidez y precisión en las operaciones, y los 2 modelos de servo con que se trabaja cumplen tanto para 100Hz como para 50Hz, que son los valores más interesantes dentro del rango.

Continuando con el análisis del código, en la declaración de variables hay que destacar las nuevas variables creadas. Estas variables tienen por objetivo almacenar valores de ángulo de las articulaciones de forma iterativa. También se crean variables para calcular los tiempos que se estima que tardará cada servomotor en ejecutar su operación y otras variables auxiliares necesarias para almacenar el número de operaciones.

Todas estas variables se complementan con las variables contador que han sido declaradas como variables globales al principio del programa. No se reproducen de nuevo porque no es necesario.

Entrando al contenido del bucle principal, puede comprobarse que ya no hay solo un “if”, sino 2. Esto se debe a que, como para calcular incrementos de posición iterativos para los servos, es necesario contar con el valor de partida y el valor de referencia del ángulo, pero los servomotores no adquieren una posición y la mantienen hasta que no se les indica con una señal eléctrica, es necesario llevarlos a una posición inicial para tener un punto de partida universal.

Además, con una variable booleana de “primera vez”, se asegura que tan solo se adquiere la posición inicial una vez por ejecución del nodo:

```
if(primeravez==1){
    printf("Inicializando\n");
    q1g=0;
    q2g=0;
    q3g=90; //0+90
    primeravez=0;
    refq1=q1g;
    refq2=q2g;
    refq3=q3g;
    q1g0=q1g;
    q2g0=q1g;
    q3g0=q3g;
}
```

Como puede comprobarse, las instrucciones de cálculo de la posición en microsegundos, almacenado de este valor en los mensajes de los ángulos y publicación de los mismos están fuera de ambas condiciones “if”, porque son comunes a ambas y porque, como se ha indicado previamente, de lo contrario la publicación no funcionaría bien.

En cuanto al segundo “if”, al que solo se entra cuando se tienen todas las coordenadas y el tiempo, se observa cómo ha cambiado significativamente parte del funcionamiento de los cálculos de la referencia para los servomotores:

En primer lugar, tras almacenar los valores de las coordenadas de referencia recibidas por suscripción, se procede a estimar el tiempo que tardará cada servomotor en ejecutar su operación si la realiza de golpe. Esta estimación se hace mediante el parámetro indicado por el fabricante “segundos/60”:

```
if(q1g>q1g0){
    t1=(q1g-q1g0)*0.14/60;}
else{
```

```
t1=(q1g0-q1g)*0.14/60;}
tmax=t1;
if(q2g>q2g0){
t2=(q2g-q2g0)*0.14/60;}
else{
t2=(q2g0-q2g)*0.14/60;}
if(t2>tmax){
    tmax=t2;}
if(q3g>q3g0){
t3=(q3g-q3g0)*0.22/60;}
else{
t3=(q3g0-q3g)*0.22/60;}
if(t3>tmax){
    tmax=t3;}
if(tmax<vart && tmax>0){
    tmax=vart;
}
```

De este fragmento de código destaca que se distingue los casos en que la referencia tiene un valor numérico mayor o menor que el valor de partida. Se hace así para evitar el uso de la función valor absoluto, “abs(...)”, porque en el habitual caso de que tanto el ángulo de partida como el de referencia sean el mismo (por ejemplo si se le indican las coordenadas en que ya se encuentra), la función abs(0) daría como resultado una indeterminación (*Nan, not a number*).

Se toma como tiempo el máximo entre el mayor de los tiempos estimados de los servomotores y el tiempo recibido por mensaje. Además, si el tiempo estimado es 0, porque la posición de partida y de llegada son la misma, no se toma el valor recibido por mensaje, porque, como se verá a continuación, en caso de seguir adelante se estaría ordenando al programa que tardase un tiempo determinado en realizar incrementos en la referencia de posición de 0°.

Prosiguiendo con el código, se calcula un número de iteraciones común a los tres servomotores, de forma que se asegura que los tres lleguen a la vez a su posición final. La coincidencia entre número de iteraciones ejecutadas y tiempo total con las que se calculan se da, como se ha indicado, gracias a la coincidencia entre frecuencia del bucle y frecuencia de muestreo:

```
n_it=tmax/tm;
```

Con este número de iteraciones se plantea una disyunción mediante condiciones “if”. En caso en que la posición de referencia sea la misma que la posición de partida (n_it=0), el programa deja de calcular, y reinicia las booleanas de las coordenadas, de modo que sigue publicándose el valor actual de posición, a la espera de unas coordenadas diferentes a las que ya se tienen:

```
else if(n_it==0){
    i=0;
    visto_x=0;
    visto_y=0;
    visto_z=0;
```

```
visto_t=0;  
}
```

En caso contrario, es decir, en el habitual caso de que las coordenadas exigidas sean diferentes a las que ya se tienen, habrá resultado un número de iteraciones superior a 1 iteración ($n_it > 1$), y para ese caso se calculan, de entrada, los incrementos graduales que harán falta en cada servomotor para que lleguen sincrónicamente a su posición final:

```
if(n_it>=1){  
  incre1=(q1g-q1g0)/n_it;  
  incre2=(q2g-q2g0)/n_it;  
  incre3=(q3g-q3g0)/n_it;
```

Seguidamente, se calculan referencias de posición acumuladas. Como no puede hacerse un bucle “for” o “while” porque no puede repetirse la operación de publicación dentro de un mismo bucle principal (como ya se ha explicado), se utiliza un contador (i) que se incrementa tras cada cálculo de una nueva referencia acumulada, de forma que lleva la cuenta de las iteraciones que se han realizado hasta cumplirlas todas. Cuando este contador alcanza (y supera) el valor del número de iteraciones, se entra en una segunda condición “if” que reinicia el contador i, reinicia las booleanas de las coordenadas y el tiempo y actualiza el valor del ángulo de partida (qg0) con el que se acaba de alcanzar (qg):

```
if(i<=n_it){  
  
  refq1=(incre1*i)+q1g0;  
  
  refq2=(incre2*i)+q2g0;  
  
  refq3=(incre3*i)+q3g0;  
  
  i++;  
}  
if(i>n_it){  
  i=0;  
  visto_x=0;  
  visto_y=0;  
  visto_z=0;  
  visto_t=0;  
  q1g0=q1g;  
  q2g0=q2g;  
  q3g0=q3g;  
}
```

Cabe reparar en que, como los servomotores trabajan con posiciones absolutas, no con incrementos, las referencias tienen que ser acumulativas desde su punto de partida:

```
refq1=(incre1*i)+q1g0;
```

A partir de este punto, el programa continúa igual que brazo_def, con la salvedad de que ahora calcula la posición en microsegundos para cada referencia incremental, y no solo para el punto final.

3.3.2.2.3- trayectoria_lineal_def:

```
#include "ros/ros.h"
    [...]
bool primeravez=1, t_adq=0, listo=1;
int i=0, k=0,j;
void prueba2Callback(const std_msgs::Float32::ConstPtr& msg)
{
    if(listo==1){
        ROS_INFO("px: [%f]", msg->data);
        px=msg->data;
        visto_x=1;}
    if(listo==0){
        ROS_INFO("Espere a que termine para indicar nuevas
coordenadas");}
}
    [...]
int main(int argc, char **argv)
{
    ros::init(argc, argv, "trayectoria_lineal_def");
    ros::NodeHandle n;
    [...]
float px0,py0,pz0,q1_rad,q2_rad,q3_rad;
    [...]
float pxt[5], pyt[5], pzt[5];
while (ros::ok())
    {
if(primeravez==1){
    [...]
    q1_rad=(3.1416/180)*q1g0;
    q2_rad=(3.1416/180)*q2g0;
    q3_rad=(3.1416/180)*(q3g0-90);
    px0=a3*cos(q1_rad)*cos(q2_rad)*cos(q3_rad)-
a3*cos(q1_rad)*sin(q2_rad)*sin(q3_rad)+a2*cos(q1_rad)*cos(q2_rad
);
    py0=a3*sin(q1_rad)*cos(q2_rad)*cos(q3_rad)-
a3*sin(q1_rad)*sin(q2_rad)*sin(q3_rad)+a2*sin(q1_rad)*cos(q2_rad
);
    pz0=a3*sin(q2_rad)*cos(q3_rad)+a3*cos(q2_rad)*sin(q3_rad)+a
2*sin(q2_rad)+d1;
    printf("px0 inicial: [%f]\n py0 inicial: [%f]\n pz0
inicial: [%f]\n", px0, py0,pz0);
    }
}
```

```
if(visto_x & visto_y & visto_z & visto_t){

    if(listo==1){
        for(j=1;j<=5;j++){
            pxt[j]=(px-px0)*(j/5.0)+px0;
            pyt[j]=(py-py0)*(j/5.0)+py0;
            pzt[j]=(pz-pz0)*(j/5.0)+pz0;
            printf("%d : x %f\n",j, pxt[j]);
            printf("%d : y %f\n",j, pyt[j]);
            printf("%d : z %f\n",j, pzt[j]);
        }
        listo=0;
        k=1;
    }

    vart=tiempo;
    var1=pxt[k];
    var2=pyt[k];
    var3=pzt[k]-d1;
    [...]
    if(tmax<vart && tmax>0){
        tmax=vart/5.0;
    }
    [...]
}
if(i>n_it){
    i=0;
    q1g0=q1g;
    q2g0=q2g;
    q3g0=q3g;
    if(k<=5){
        k++;
        printf("%d\n",k);
    }
    if(k>5){
        visto_x=0;
        visto_y=0;
        visto_z=0;
        visto_t=0;
        px0=px;
        py0=py;
        pz0=pz;
        listo=1;
        printf("listo\n");
    }
}
}
else if(n_it==0){
```

```
        i=0;
        visto_x=0;
        visto_y=0;
        visto_z=0;
        visto_t=0;

        }
}

```

[...]

Tan solo se comentarán en profundidad y reproduciendo el código para analizar mejor la parte referente al cálculo de trayectoria por puntos.

En cuanto a las variables globales tan solo se introduce una booleana (“listo”), que se incluye en las funciones “CallBack” de gestión de la suscripción para evitar que se puedan cambiar las coordenadas de referencia hasta que no se haya terminado la operación de desplazamiento. También se incluyen dos nuevos contadores, j y k. Con j se calcularán los puntos de interpolación lineal para que el elemento terminal se mueva de la posición inicial a la final siguiendo una línea recta y con k se accede a esos puntos para asegurar trayectorias angulares sincronizadas entre esos puntos intermedios.

Cambiando el número de j y k, se elige el número de puntos intermedios (en este caso, son 5 puntos).

Una vez en el bucle principal, la primera vez que se entra se indica una posición inicial y, además, se aprovecha el problema cinemático directo para tener unas coordenadas de partida:

```
px0=a3*cos(q1_rad)*cos(q2_rad)*cos(q3_rad)-
a3*cos(q1_rad)*sin(q2_rad)*sin(q3_rad)+a2*cos(q1_rad)*cos(q2_rad
);

```

Una vez se tienen nuevas coordenadas, se realiza el cálculo de los j puntos de trayectoria rectilínea, y se almacenan en vectores para px, py y pz. Se comienza desde el punto j=1, porque j=0 corresponde al mismo punto de partida, y se malgasta capacidad de cálculo:

```
if(listo==1){
    for(j=1;j<=5;j++){
        pxt[j]=(px-px0)*(j/5.0)+px0;
        [...]
    }
    listo=0;
    k=1;
}

```

Una vez hecho esto, para cada punto desde k=1 hasta el último, se calculan y publican las referencias angulares, asegurando la sincronía en la trayectoria entre puntos intermedios. El tiempo entre cada punto es el tiempo total indicado entre el número de puntos.

3.3.3- Instalación y uso del protocolo roserial:

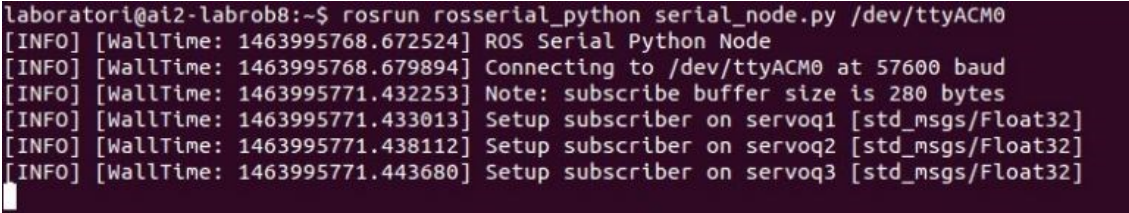
La instalación de roserial, en particular del paquete específico para la comunicación con Arduino, es sencilla y no trae consigo ningún problema. Se realiza automáticamente mediante las instrucciones desde terminal:

```
sudo apt-get install ros-indigo-roserial-arduino
sudo apt-get install ros-indigo-roserial
```

Una vez concluye la instalación, se consiguen una serie de librerías (*ros_lib*) que es necesario instalar en el entorno de Arduino, tal y como se indica en el primer tutorial de ros.org para roserial_arduino [23].

En cuanto a su uso, roserial se utiliza como un paquete más, sin modificarlo, de forma que se llama al nodo *serial_node.py* del paquete *roserial_python* y asociándole el puerto USB en que esté conectada la placa Arduino.

Si el puerto está correctamente referenciado, el nodo informará de cuáles son los *topics* por los que se comunica el programa cargado en la Arduino y qué tipo de mensajes espera recibir en caso de ser una suscripción (Figura 3.8):



```
laboratori@ai2-labrob8:~$ rosrund serial_node.py /dev/ttyACM0
[INFO] [WallTime: 1463995768.672524] ROS Serial Python Node
[INFO] [WallTime: 1463995768.679894] Connecting to /dev/ttyACM0 at 57600 baud
[INFO] [WallTime: 1463995771.432253] Note: subscribe buffer size is 280 bytes
[INFO] [WallTime: 1463995771.433013] Setup subscriber on servoq1 [std_msgs/Float32]
[INFO] [WallTime: 1463995771.438112] Setup subscriber on servoq2 [std_msgs/Float32]
[INFO] [WallTime: 1463995771.443680] Setup subscriber on servoq3 [std_msgs/Float32]
```

Figura 3.8 Terminal de Ubuntu correspondiente a *roserial*

Es importante que:

-El *ROS master* esté funcionando (es decir, que se haya realizado la instrucción *roscore*).

-La placa Arduino tenga el programa correcto cargado. En ocasiones puede haber un error en caso de haber conectado y desconectado la Arduino muchas veces sin volver a descargar el programa en ella. Rosserial informará por pantalla de un error al tratar de conectar, sugiriendo que puede haber un desfase en las versiones de roserial frente a la de Arduino, pero esto no tiene por qué ser el motivo.

Este error también se dará si, con roserial tratando de conectar, se vuelve a descargar el programa a la Arduino.

-No estén funcionando a la vez muchos programas que utilizan *drivers* para comunicarse con dispositivos periféricos a los que el ordenador se conecta mediante USB. En este caso, no es difícil que roserial no llegue a conectarse.

-Las conexiones con la placa Arduino estén correctamente realizadas. Si, por ejemplo, la demanda eléctrica a los puertos de voltaje positivo de la placa Arduino es demasiado alta, o se conectan y desconectan muchas veces los puertos de señal PWM, u otros problemas que afecten a la Arduino, roserial podría informar del error explicado anteriormente.

Cumpliendo todas las condiciones y teniendo en marcha el nodo, publicando o suscribiéndose a los *topic*, los mensajes irán del programa que opere en el ordenador (o

desde la terminal donde se publique o suscriba) hasta Arduino y viceversa con tan solo utilizar las instrucciones usuales.

3.3.4- Implementación y uso global de los programas:

Para utilizar los programas simplemente hay que encender el *ROS master*, utilizar la instrucción *roslaunch* para poner en marcha tanto el programa que se desea ejecutar de los tres disponibles como *rosserial*. Todo ello ha de realizarse con la placa Arduino conectada al ordenador, con su puerto USB correctamente referenciado y con el programa correspondiente descargado.

Una vez hecho esto, desde un mismo terminal hay que indicar las coordenadas x, y, z que servirán de objetivo y, en el caso de *brazo_def_sinc* y de *trayectoria_lineal_def*, también el tiempo en que se deben ejecutar las operaciones.

Cada coordenada y el tiempo corresponden a un mensaje distinto, que se indica en la terminal, con la instrucción *rostopic pub*, el *topic*, el tipo de mensaje y el valor de la coordenada. Para indicar un valor negativo, es necesario escribir "--" antes del valor negativo (-- -0.4). En la figura 3.9 puede verse un ejemplo de publicación:



```
laboratori@ai2-labrob8:~$ rostopic pub teclado3 std_msgs/Float32 0.4
publishing and latching message. Press ctrl-C to terminate
```

Figura 3.9 Ejemplo de publicación de coordenada. Coordenada z=0.4 m.

También podría haberse implementado incluyendo todas las coordenadas en un mismo mensaje, pero de esta forma es más sencillo corregir una de las coordenadas en caso de que por error se introduzca una no deseada. En ese caso, mientras no se hayan indicado aún todas las coordenadas, basta con repetir la operación de publicación en el mismo *topic* y variando el valor.

Una vez terminadas las operaciones, se indican nuevas coordenadas y tiempo de la misma manera, y el brazo robótico las ejecuta.

3.3.5- Particularidades del trabajo con Raspberry Pi:

En primer lugar, se instala Raspbian Jessie en la Raspberry Pi, instalando en una tarjeta micro SD la imagen virtual del sistema operativo. Se ha hecho desde Windows, puesto que para éste sistema operativo hay un asistente de instalación que facilita el proceso.

Para instalar ROS Hydro, que es compatible con *rosserial* para Linux empotrado, se han seguido unas instrucciones basadas en las de *ros.org* pero completadas con indicaciones importantes [30]. La particularidad en este caso es que todo lo fundamental para que ROS funcione se instala en un *workspace*, *ros_catkin_ws*, mientras que los distintos paquetes, protocolos, librerías...se instalan en otro *workspace*, *catkin_ws*.

La instalación de *rosserial_embeddedlinux* también se realiza siguiendo dichas instrucciones, y se instala el protocolo como directorio con múltiples paquetes, que incluyen *rosserial_arduino*.

Debido a estas particularidades, aunque la creación y construcción de paquetes se ordena igual que en ROS en un ordenador, trae consigo muchos directorios dentro de un mismo *workspace* que tienen el mismo nombre por defecto, lo que lleva a que tanto en los documentos *CMakeLists.txt* como al escribir instrucciones como *roslaunch* que hacen

referencia a nodos de determinados paquetes, haya que explicitar toda la ruta hasta el archivo o directorio referenciado.

La comunicación con Arduino una vez instalado rosserial funciona exactamente como en un PC, desde referenciar el puerto USB hasta poner en marcha el nodo *serial_node.py*, con la salvedad explicada en el párrafo anterior. Para aligerar la aplicación, la IDE de Arduino no se instala en la Raspberry, y el programa de la placa Arduino se descarga desde el PC y luego se utiliza en la conexión con la Raspberry.

Pese a ser funcional, la comunicación mediante publicación y suscripción con Arduino y rosserial es lenta en Raspberry y consume mucha capacidad de procesador, de modo que se estudia para futuros proyectos la implementación con un módulo para Raspberry Pi, enfocado al control de motores “paso a paso”, Gertbot [31].

Como el trabajo se ha desarrollado con servomotores, no compatibles con Gertbot, no se ha implementado a nivel práctico, pero sí se han resuelto los problemas informáticos que presentaban los drivers de Gertbot con ROS, convirtiendo los archivos .c de dichos drivers en librerías .h para incluir en el nodo diseñado para ROS.

3.4- Rediseño y ensamblaje del brazo robótico:

El presente trabajo ha partido de un modelo de brazo articulado pequeño que, tanto por dimensiones como por aplicación y problemas con algunas piezas, ha sido rediseñado en múltiples aspectos, como se verá a continuación.

3.4.1- Análisis del diseño original:

El diseño de partida es un brazo robot de 6 grados de libertad, de bajo coste de impresión y dimensiones reducidas, y ha sido descargado gratuitamente de la página web Thingiverse [32], donde su creador lo ha puesto a disposición del usuario.

El diseño ha sido realizado mediante programas de diseño de código libre, como OpenScad, que será también el programa que se utilice para el rediseño del brazo.

Para entender correctamente los cambios que hay que implementar, se procede a analizar el brazo en su forma de partida (Figura 3.10).

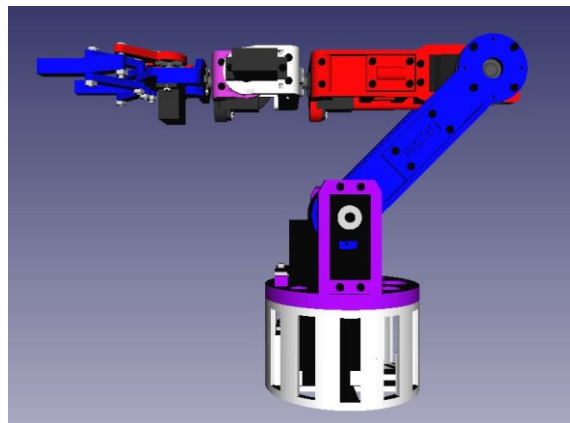
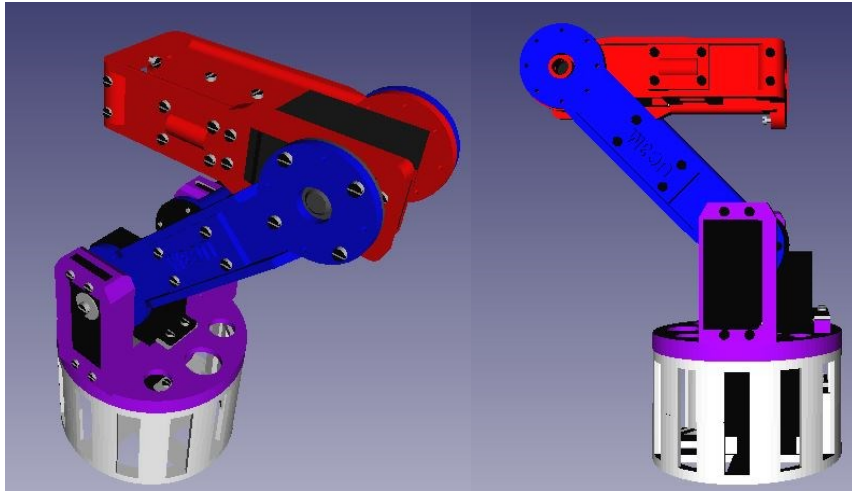


Figura 3.10 Diseño de partida, de perfil.

Mediante el programa de diseño y representación tridimensional de código libre, FreeCAD, se consigue una imagen completa del diseño del brazo completo original.

Dado que no se va a trabajar con 6 grados de libertad, sino tan solo con 3, se ocultan los eslabones correspondientes a la pinza y la muñeca, ganando visibilidad (Figuras 3.11 y 3.12).

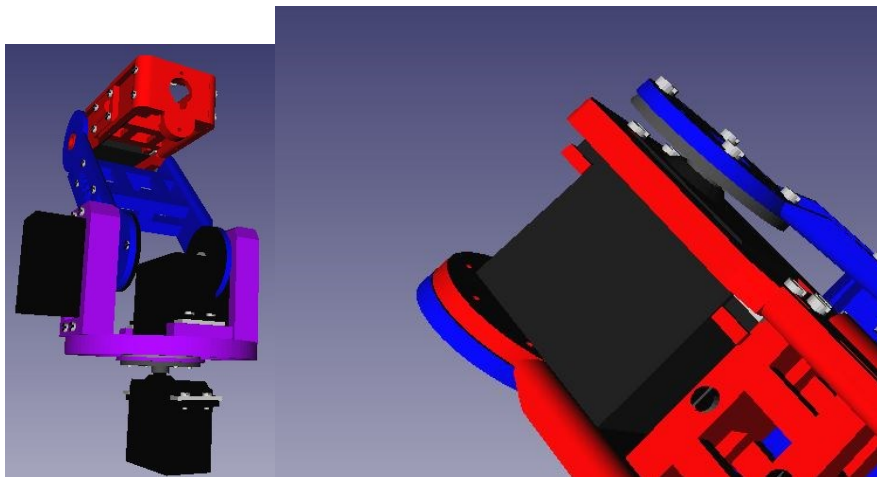


Figuras 3.11 y 3.12 Brazo original sin pinza

En las imágenes anteriores, ya sin la pinza, puede observarse sin dificultad, con ayuda de la herramienta de medida de FreeCAD, que los eslabones 2 y 3 (azul y rojo, respectivamente), no tienen la misma longitud, y se encuentra en ambos casos en el orden de los 10 cm.

Es conveniente recordar que el brazo va a anclarse sobre la cubierta superior de un robot móvil Summit, de modo que, para poder llegar a los objetos que se desee manipular, estos eslabones deberán tener una longitud del orden de los 40 cm.

Continuando con el análisis, es necesario valorar los servomotores que utiliza el diseño de partida:



Figuras 3.13 y 3.14 Detalles de servomotores originales

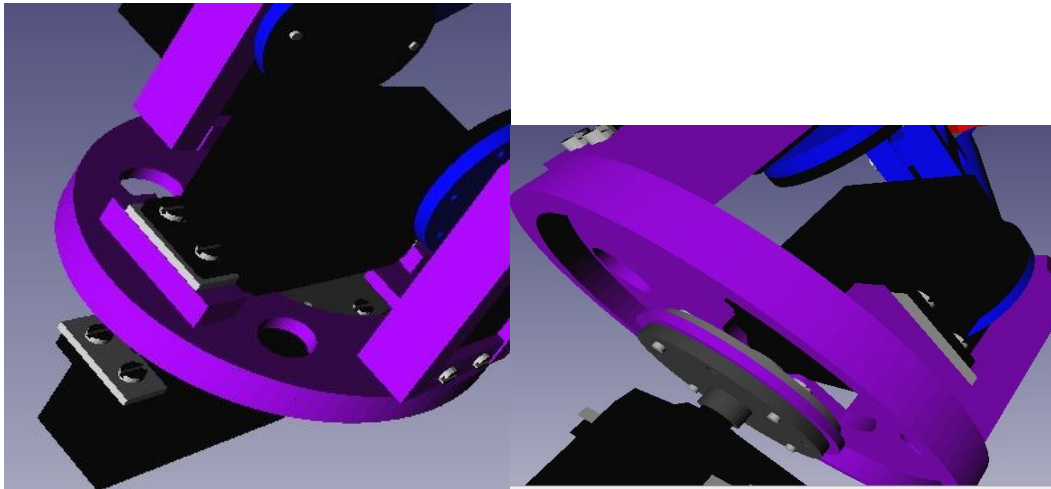
Como se observa en las figuras 3.13 y 3.14, una vez reducidos a 3 los grados de libertad del brazo articulado, quedan 4 servomotores. Se ha desactivado la visibilidad del eslabón 0 para que sea más cómoda la visualización de dichos servomotores.

Dos servomotores de tipo Futaba s3003 permiten el giro en la articulación correspondiente a q_1 .

Otros dos servomotores, uno Futaba s3004 y otro Futaba s3003, permiten los giros de las articulaciones correspondientes a q_2 y q_3 , respectivamente.

La información sobre el tipo de servomotor viene incluida como nombre de su objeto de representación tridimensional en el archivo FreeCAD proporcionado por el creador del diseño original del brazo, y buscando en el catálogo de Servodatabase.com, se comprueba que el par mecánico proporcionado por ambos servomotores es de entre 3.17 y 4.10 kg·cm [33][34], a todas luces insuficiente en comparación a los valores de entre 10 y 25 kg·cm que se requerirían en el presente trabajo.

Precisamente este reducido par mecánico proporcionado por los servomotores del diseño original, hace que dicho diseño requiera dos de ellos para realizar el giro correspondiente a q_1 , viéndose obligado su creador a recurrir a una complicada configuración que ancla uno de los servomotores al eslabón 0 (blanco) y otro al 1 (morado), para que la acción resultante del giro de ambos logre mover el brazo.



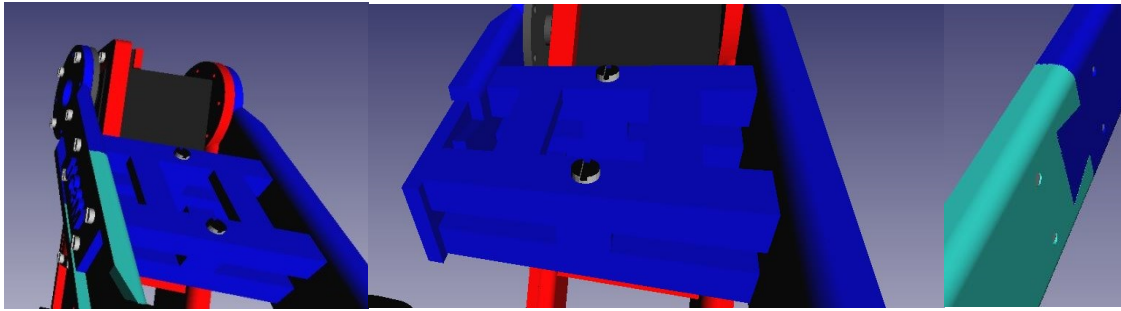
Figuras 3.15 y 3.16 Detalles de articulación q_1

En las figuras 3.15 y 3.16 puede observarse con bastante claridad esta configuración. Como puede verse, el nexo de unión entre ambos servos es una arandela (morada, en esta representación), a la cual se han anclado mediante tornillos y sus platos.

En el diseño del presente trabajo, gracias al uso de servos que proporcionan un par significativamente mayor (del orden de 5 veces más que los del diseño original), puede prescindirse de uno de los dos servos y simplificar la configuración de esta articulación.

Por último en cuanto al diseño de partida, existe un error en el diseño de una de las piezas, que hace que un refuerzo utilizado en el eslabón 2 (azul) atraviese virtualmente a los elementos constituyentes fundamentales de dicho eslabón, sin que esos elementos constituyentes tengan ningún agujero pasante practicado para que sea posible esa configuración.

Es decir, el refuerzo es demasiado grande y da lugar a un montaje que es físicamente imposible, como se muestra en las figuras 3.17, 3.18 y 3.19, donde se ha coloreado de un azul más claro uno de los elementos constituyentes fundamentales del eslabón 2 para que se pueda apreciar el problema con más claridad:



Figuras 3.17, 3.18 y 3.19 Detalle error de diseño original

Para mostrarlo con claridad, se oculta en la figura 3.18 el elemento coloreado en azul claro, de modo que puede apreciarse cómo el refuerzo está ocupando su espacio, y en la figura 3.19 se oculta el refuerzo y se vuelve a mostrar el elemento de color azul claro, comprobándose que no tiene agujeros pasantes para que sea posible que el refuerzo lo atraviese.

A continuación, se explican las soluciones que se ha dado a estas situaciones.

3.4.2- Rediseño inicial:

El rediseño del brazo se ha realizado a partir de piezas determinadas del brazo de partida, utilizando software de diseño de código libre, concretamente OpenSCAD. La única excepción a esto es la solución al problema con la pieza de refuerzo del eslabón 2, donde se ha recurrido a otro programa de diseño virtual tridimensional: NX de Siemens, con ayuda de un técnico del Departamento de Ingeniería de Sistemas y Automática de UPV.

Sin entrar en detalles sobre la forma de trabajar con OpenSCAD, sí existen una serie de factores a tener en cuenta en relación a la forma de trabajar con dicho programa, dado que han condicionado la forma en que se han enfocado muchas de las operaciones de rediseño:

-En OpenSCAD ha de trabajarse directamente con código informático, introduciendo formas geométricas (cilindros, cubos y, en algunas ocasiones, otros prismas) y realizando operaciones de diferencia, simetría, distribución en matriz polar...entre dichas formas geométricas básicas. Ello implica que una pieza con chaflanes, agujeros no pasantes, raíles, etc...puede presentar un código con un alto grado de complejidad.

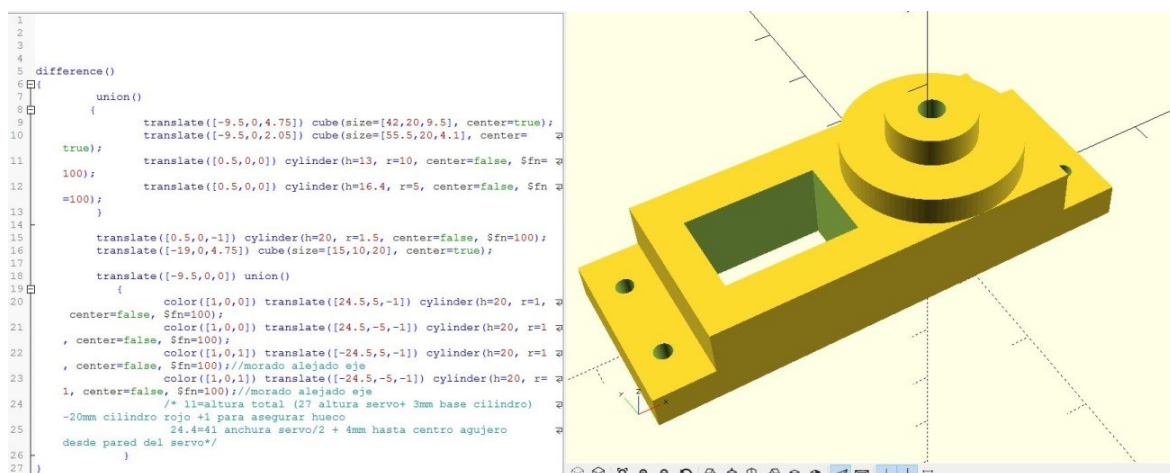


Figura 3.20 Muestra de funcionamiento de OpenSCAD

-El programa no permite observar los cambios que se escriben en forma de código a la vez que se van implementando, sino que debe realizarse un *render* de la pieza a cada modificación. Esto dificulta mucho la tarea de comparación entre el estado original y el modificado, ya que la pre visualización que ofrece el programa es bastante pobre, como se observa en la figura 3.21:

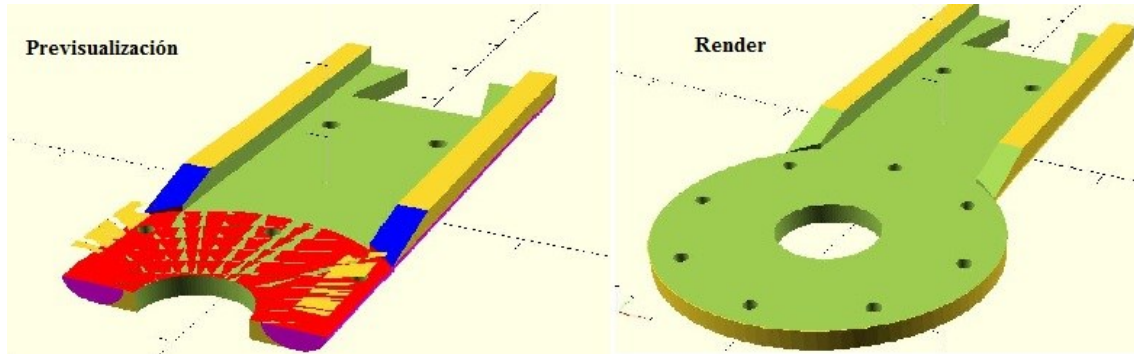


Figura 3.21 Muestra de inconvenientes de pre visualización en OpenSCAD

-En OpenSCAD la ventana que muestra la figura permite orientarla y ampliar la imagen, pero no es posible seleccionar elementos u operaciones de una pieza (como agujeros, matrices de elementos, prismas...) y nombrarlos, de forma que, salvo que el creador original haya indicado mediante comentarios junto al código qué realiza cada línea de código informático y sobre qué operación o conjunto de operaciones trabaja, resulta altamente complejo averiguarlo, siendo forzoso a menudo recurrir al ensayo-error borrando ciertas líneas de código para comprobar el resultado.

-Al haber de trabajar desde código, OpenSCAD implementa un sistema de referencia global para cada pieza, de modo que para que sea más sencillo posicionar cada una de las formas geométricas primitivas que combinamos para dar lugar a la pieza, suele activarse la opción que refiere las coordenadas globales de cada forma primitiva al centro geométrico de dicha forma. Esto implica que el tamaño de cada forma en cada una de las dimensiones del espacio se dará simétricamente respecto a su centro geométrico. Ello ha de tenerse muy en cuenta a la hora de cambiar la longitud de una pieza, como se podrá comprobar a continuación.

-OpenSCAD no cuenta con una herramienta de medida, pero sí incluye una escala milimétrica sobre el sistema de referencia global, que permite medir con una precisión razonable para los objetivos de este trabajo.

3.4.2.1- Redimensionado de los eslabones 2 y 3:

Para redimensionar el eslabón 2, los elementos constituyentes fundamentales son las piezas llamadas “eslabón 2A”. Hay 4 de estas piezas, 2 a cada lado del brazo y conectadas entre sí, que son iguales en forma y dimensión salvo por el extremo, en que puede elegirse si es saliente o entrante (cambiando el valor de una variable binaria en el código), de modo que pueden conectarse unas con otras, como se muestra en la figura 3.22.

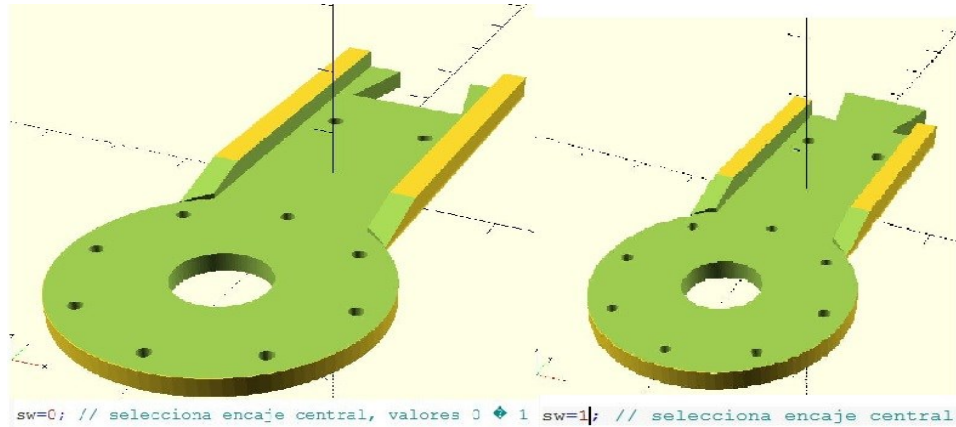


Figura 3.22 Detalle terminación pieza eslabón 2A

De acuerdo con las limitaciones que establece el propio programa, derivadas de los factores que se han explicado anteriormente, el redimensionado tiene, como particularidad, que ha debido compensar el crecimiento de las diferentes formas geométricas primitivas con un desplazamiento de su centro proporcionado.

Es decir, dado que las formas incrementan sus dimensiones simétricamente respecto de su centro, ha sido necesario desplazar también ese centro exactamente la mitad del crecimiento incorporado. Para mantener el encaje o saliente del final de la pieza, así como los agujeros y los acabados en rampa, también ha de preverse un desplazamiento proporcionado de las formas geométricas con las que se componen estos elementos y operaciones.

El resultado final se representa en la figura 3.23:

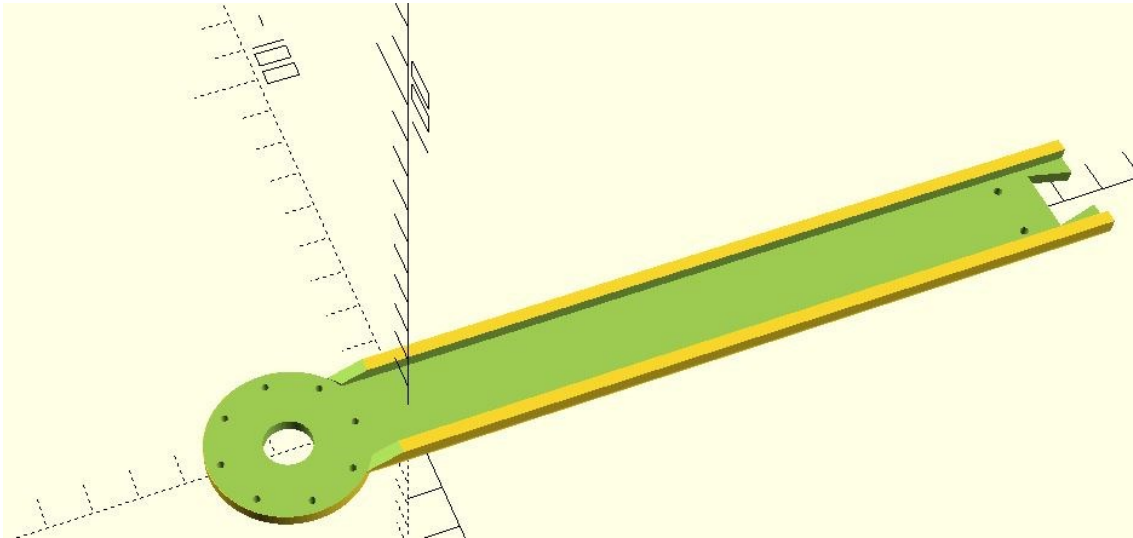


Figura 3.23 Resultado final del alargamiento de la pieza eslabón 2A

A continuación, se adjunta una imagen con el código de OpenSCAD (Figura 3.24), con el incremento DI y su implementación a la hora de alargar algunas formas geométricas y de desplazar elementos, operaciones, y el centro de esas mismas formas geométricas para compensar el crecimiento simétrico respecto a su centro.

```
sw=0; // selecciona encaje central, valores 0 1
DI = 132;
difference()
{
  union()
  {
    translate([0,2.5+DI/2,4.5]) cube(size=[25,59+DI,3], center=true);
    translate([0,2.5+DI/2,3]) cube(size=[19,59+DI,6], center=true);
    color([1,0,0]) rotate ([0,0,0]) translate([0,-27,0]) cylinder(h=3, r=18, center=false, $fn=100);
    color([1,0,1]) rotate ([90,0,0]) translate([9.5,3,-2.48-DI/2]) cylinder(h=59+DI, r=3, center=true, $fn=100); // lateral redondo
    color([1,0,1]) rotate ([90,0,0]) translate([-9.5,3,-2.48-DI/2]) cylinder(h=59+DI, r=3, center=true, $fn=100); // lateral redondo
  }
  translate([0,-27,-7.5]) cylinder(h=20, r=5.4, center=false, $fn=100);
  translate([0,-27,-7.5])
  {
    for ( i=[0:7]) //genera los refuerzos donde no encaja la parte superior (las dos lineas)
    {
      rotate (i*360/8)
      translate([15,0,0]) cylinder(h=20, r=0.8, center=false, $fn=100);
    }
  }
  translate([0,2.5+DI/2,7]) cube(size=[18.9,70+DI,8], center=true); // crea refuerzo lateral
  color([1,0,0]) rotate ([0,0,0]) translate([0,-26.9,3]) cylinder(h=3, r=18, center=false, $fn=100); // refuerzo lateral
  if(sw==0)
  {
    linear_extrude(height = 20, center = true, convexity = 10, twist = 0)
    rotate([180,0,0])translate([-12.5,-32-DI,0])
    polygon([[7.5,0],[5,10],[20,10],[17.5,0]]);
  }
  if(sw==1)
  {
    linear_extrude(height = 20, center = true, convexity = 10, twist = 0)
    translate([-12.5,22+DI,0])
    polygon([[7.5,0],[5,10],[20,10],[17.5,0],[25,0],[25,10],[0,10],[0,0]]);
  }
  translate([6,13.5+DI,0]) cylinder(h=20, r=1, center=false, $fn=100); // sujeticn central
  translate([-6,13.5+DI,0]) cylinder(h=20, r=1, center=false, $fn=100); //sujeci central
  color([0,0,1]) translate([0,-14,3]) rotate ([20,0,0]) translate([0,5,5]) cube(size=[50,10,10], center=true);//recorta pico lateral
}
```

Figura 3.24 Ejemplo código definitivo para el alargamiento de la pieza eslabón 2A

Para redimensionar el eslabón 3, la opción más sencilla y eficaz es redimensionar las 2 piezas simétricas llamadas eslabón 3C, con un enfoque y código análogo al de la pieza “eslabón 2A”. Proceder de esta forma permite no alterar las piezas “eslabón 3A” y “eslabón 3B”, que tienen formas y operaciones realizadas más complicadas dado que incorporan espacio y lugar de anclaje para el servomotor.

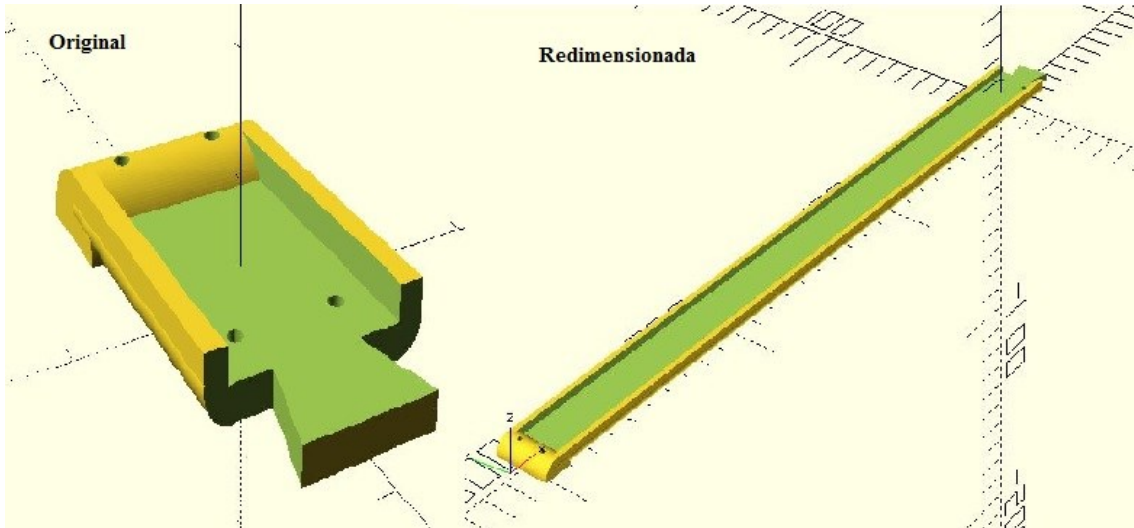
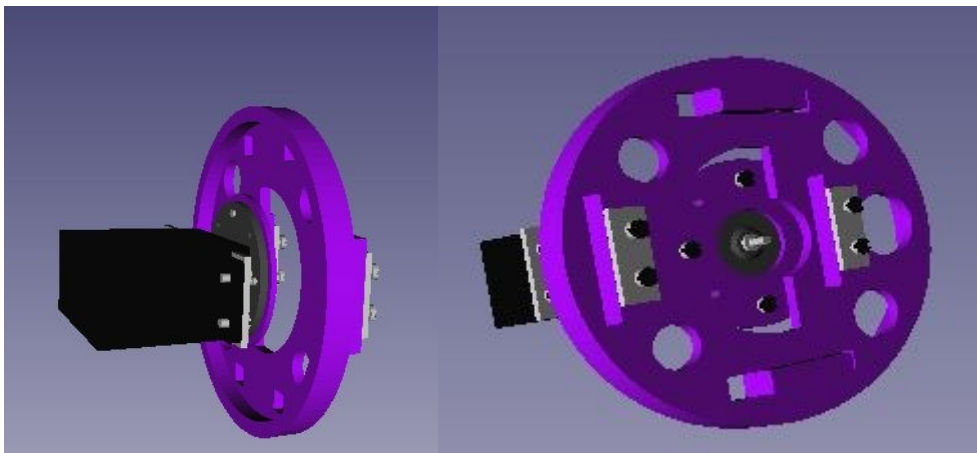


Figura 3.24 Redimensionado de la pieza eslabón 3B

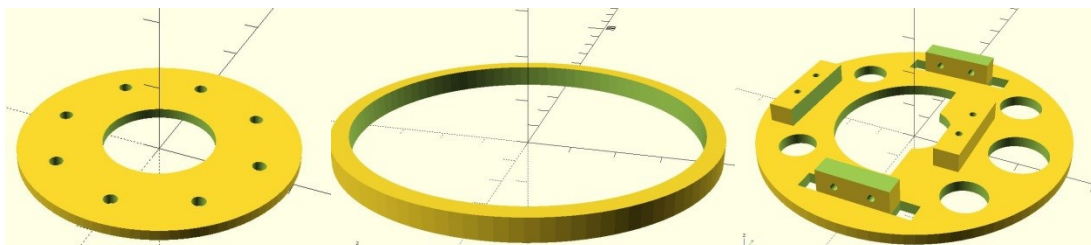
3.4.2.2- Articulación correspondiente a q_1 :

Al prescindir de uno de los dos servomotores, hay que conectar el eje del servomotor anclado al eslabón 0 (blanco) al eslabón 1 (morado), de forma que al girar, haga girar consigo a todo el brazo a partir del eslabón 1, funcionando como cadera.

Para comprender la solución de diseño implementada, conviene ver antes las piezas que originalmente transmitían el giro (Figuras 3.25 y 3.26):



Figuras 3.25, 3.26 Detalle eslabón 1 original



Figuras 3.27, 3.28, 3.29 Detalle eslabón 1 desglosado en piezas

Como se puede comprobar en las imágenes, hay 3 piezas fundamentales: la pequeña arandela llamada eslabón 1D (Figura 3.27), que conecta los dos servomotores del diseño original, el anillo llamado eslabón 1C (Figura 3.28), que sirve de apoyo del eslabón 1 sobre el eslabón 0, y la pieza llamada eslabón 1A (Figura 3.29), donde se ancla el servomotor superior.

La pieza eslabón 1A tiene orificios para, según los comentarios adjuntos al código de dicha pieza, ahorrar material y, además, el agujero central para permitir la comunicación de ambos servos.

Para comunicar el servomotor que se mantiene, el inferior, con la pieza superior (eslabón 1A), y así transmitir el giro, se unirá el servo a la pequeña arandela (eslabón 1D) y ésta deberá unirse a una pieza que a su vez se una hasta la pieza llamada eslabón 1A.

Para hacer esto posible, se convierte la pieza con forma de anillo (eslabón 1C) en una tapa maciza, a la que se atornillará la arandela (eslabón 1D), unida a su vez al servomotor inferior.

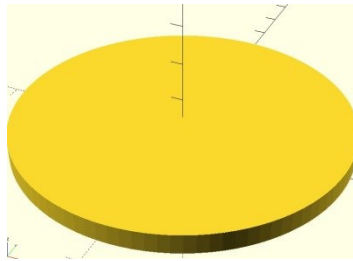


Figura 3.30 Eslabón 1C modificado

Y a continuación, hay que adaptar la pieza superior (eslabón 1A), para la nueva situación: retirar los anclajes para el servomotor superior, ampliar el orificio central para que los tornillos que unen arandela (eslabón 1D) y tapa maciza (eslabón 1C) no rocen con la pieza superior y eliminar los orificios laterales para poder atornillar la tapa maciza.

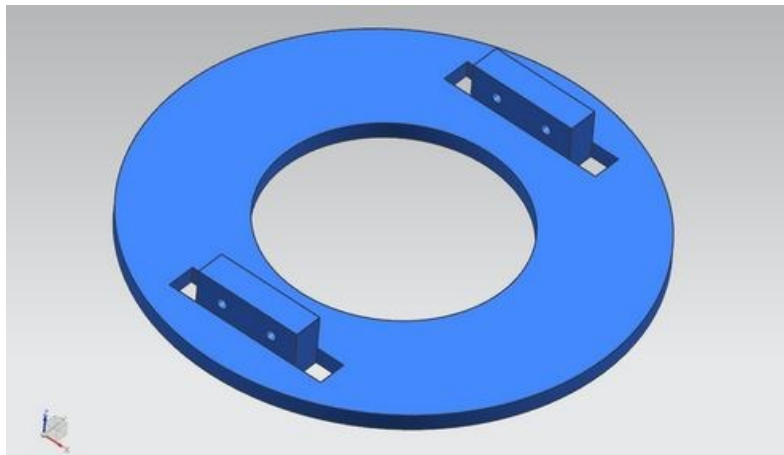


Figura 3.31 Eslabón 1A modificado

De este modo, el eslabón 0 está anclado al Summit, el servomotor inferior anclado al eslabón 0, la arandela unida al eje del servomotor, la pieza maciza unida a la arandela y la pieza superior a la pieza maciza. Se ha logrado mantener la transmisión del giro con un solo servo en la articulación correspondiente a q_1 .

Se ha decidido mantener las 3 piezas como independientes en lugar de combinarlas en una sola de mayor grosor porque de este modo se ahorra material, se facilita el taladrado y atornillado, y, en caso de un error en la impresión o incluso en el rediseño, es más sencillo solventarlo y volver a imprimir.

3.4.2.3- Error en el refuerzo del eslabón 2:

Para solucionar este problema, dadas las limitaciones de OpenSCAD a la hora de modificar piezas previamente diseñadas, y dada la sencillez constructiva del refuerzo, se opta por crear de nuevo el refuerzo con el programa de diseño NX de Siemens.

3.4.3- Impresión 3D de las piezas:

Todas las piezas de esta primera fase de rediseño se imprimen agrupándolas en 4 sesiones de impresión que ocupan una bandeja de tamaño A4 cada una. El objetivo es que las piezas medianas y pequeñas se impriman en dos sesiones nocturnas que ocupen aproximadamente 8 horas cada una, liberando la impresora para otros usos durante el día, mientras que las piezas grandes que ocupan prácticamente toda la bandeja se impriman en 2 sesiones de alrededor de 3 horas cada una, durante el día.

De esta forma se optimiza el uso de la impresora 3D, cosa que es recomendable porque la impresión suele ser un cuello de botella, condiciona todo el trabajo posterior, así que debe hacerse lo más rápido y acertadamente posible.



Figura 3.32 Bandejas 1 y 2 recién impresas



Figura 3.33 Una de las bandejas 3 recién impresa

En la línea de evitar problemas con la impresión, es recomendable rociar con laca para impresoras 3D la bandeja antes de cada impresión para que las piezas no se queden pegadas a la superficie de la bandeja. También es una buena práctica comprobar al menos una vez el estado del extrusor por el que saldrá el hilo de polímero caliente. Si es necesario limpiar el extrusor, esto puede hacerse empleando algún material de impresión consistente, como el Nylon.

Cabe explicar que se ha escogido el material PLA [35] (poliácido láctico) por su carácter altamente deformable, que lo hace idóneo para afrontar las pequeñas modificaciones y operaciones típicas de los prototipos (taladrado, vaciado...). Se descarta el material ABS [36] (acrilonitrilo butadieno estireno) porque, si bien es más resistente, duro y tenaz que el PLA, el acabado final de las piezas es menos exacto, y los salientes se desprenden con facilidad.

3.4.4- Ensamblaje, operaciones sobre las piezas y correcciones:

Para evitar la reiteración a lo largo de este apartado de la memoria del presente Trabajo Fin de Grado, se exponen a continuación una serie de consideraciones comunes a todo el proceso de ensamblaje, operaciones practicadas a las piezas y correcciones en el diseño de algunas, de manera que, después, se documentará ordenadamente este proceso deteniéndose tan solo en aquellas situaciones particulares.

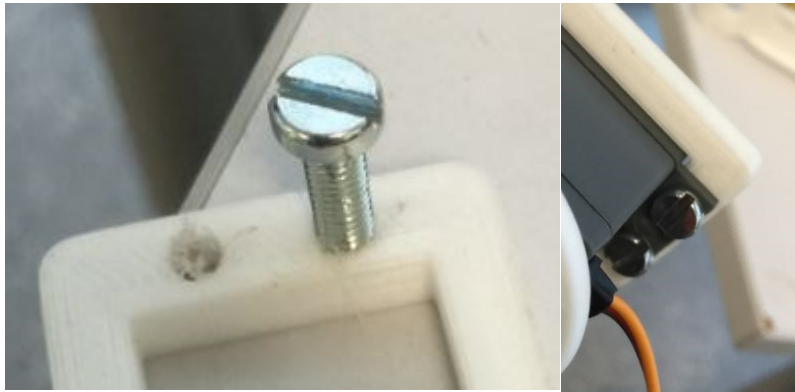
-Como se ha indicado, la impresión 3D supone un cuello de botella, así que ha de procurarse recurrir a ella solo cuando sea estrictamente necesario, tratando de valorar antes toda solución alternativa más rápida, reflexionando bien el diseño que se desea imprimir cuando es necesario e intentando enfocar siempre las piezas de forma que sean pequeñas y versátiles, para que no sean complicadas de sustituir.

-Las piezas más grandes, que son las que tardan más en imprimirse y consumen más recursos, han de cuidarse especialmente. En el presente trabajo dichas piezas (las principales de los eslabones 2 y 3) tan solo se han impreso una vez, buscando siempre alternativas cuando ha sido necesaria una modificación o corrección que pudiera afectarles.

-Tanto a la hora del rediseño como de posteriores correcciones, siempre se ha evitado incluir los agujeros para tornillos como parte del archivo de impresión. En cambio, los orificios se han taladrado en su mayoría mediante un taladro estacionario, ganando versatilidad a la hora de ensamblar piezas, en particular en el caso de los servos, y ahorrando tiempo en la fase de diseño.

-Para el ensamblaje pieza-pieza se han empleado tornillos de 2.5 mm y 3 mm de diámetro, mientras que para fijar los servos firmemente a las piezas se han usado tornillos de 4 mm de diámetro.

-El material PLA es altamente deformable, propiedad que permite un ajuste con un fuerte apriete, para asegurar la firmeza de las uniones tanto pieza-pieza como pieza-servo. En concreto, en el presente trabajo se ha proporcionado un apriete no muy acusado en las uniones pieza-pieza de las articulaciones correspondientes a q_2 y q_3 utilizando una broca de 3 mm para tornillos de 3 mm de diámetro, y un apriete muy acusado a la hora de fijar los servos utilizando una broca de 3.5 mm para tornillos de 4 mm de diámetro, como se muestra en las figuras 3.34 y 3.35.



Figuras 3.34 y 3.35 Detalle ajuste con apriete acusado para fijar servomotores

-Un inconveniente al taladrar el PLA con el taladro estacionario es que no solo se descompone en forma de virutas, sino que parte de esta viruta se reblandece por el calor debido a la fricción y se pega a la broca del taladro. Es necesario retirar esta viruta adherida a la broca, o actuará como material abrasivo al taladrar de nuevo, dando lugar a malos acabados.



Figura 3.36 Detalle viruta de PLA adherida a la broca

-Otro inconveniente al taladrar el PLA es que, como cada pieza está compuesta como una malla de fibras de polímero en que las superficies tienen un acabado más consistente que el interior, es común que la superficie por donde sale la broca al taladrar se agriete o deforme. Es importante tener esto en cuenta para alisar después la superficie, retirando los salientes indeseados sin generar demasiado calor por rozamiento, para no deformarlo más.



Figuras 3.37 y 3.38 Detalle deformación y corrección del PLA tras taladrado

-Taladrar los agujeros que sean necesarios en lugar de incluirlos en el archivo de diseño previo a impresión ahorra tiempo, como se ha dicho, pero exige particular cuidado a la hora de taladrar dos o más piezas que cuentan con agujeros coaxiales. En estos casos, es altamente recomendable encontrar medios con que fijar dichas piezas para asegurar que efectivamente los orificios son coaxiales en su disposición final, porque de lo contrario intentar hacer pasar los tornillos a través de dichos agujeros podría dañar las piezas gravemente. En la imagen a continuación se muestra un ejemplo, donde se ha garantizado esa fijación atornillando a medida que se va taladrando.

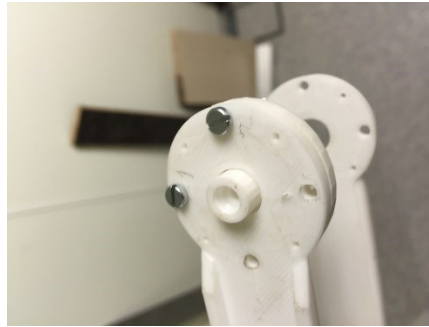


Figura 3.39 Ejemplo de alineación para garantizar coaxialidad tras el taladrado

-El ensamblaje de eslabones, que se da mediante las articulaciones, tiene una particularidad debido al funcionamiento de los servomotores. Los servomotores adquieren y mantienen la posición angular correspondiente a una determinada señal eléctrica, y dejan de mantener dicha posición cuando dejan de ser alimentados con esa señal. Ello implica que para ensamblar el brazo de forma que sea coherente con el modelo cinemático previsto, cuando se unan los eslabones en las articulaciones correspondientes, habrá que hacerlo mientras se alimenta al servo con la señal adecuada (por ejemplo, enviar a un servomotor la señal correspondiente a su posición de 0° y montar los eslabones de forma coherente a dicha posición).

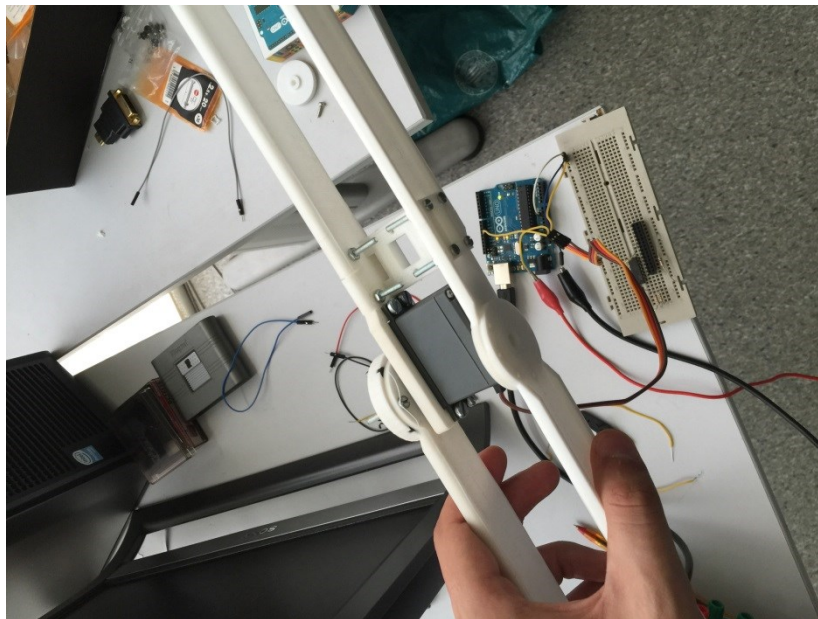


Figura 3.40 Ejemplo montaje articulación con servomotor recibiendo señal eléctrica adecuada

Expuestas estas consideraciones, ya es posible proceder a documentar el ensamblaje del brazo robótico:

En primer lugar, se han ensamblado por separado los eslabones 0, 2 y 3, y se han fijado firmemente los servos a las piezas que los sustentan:

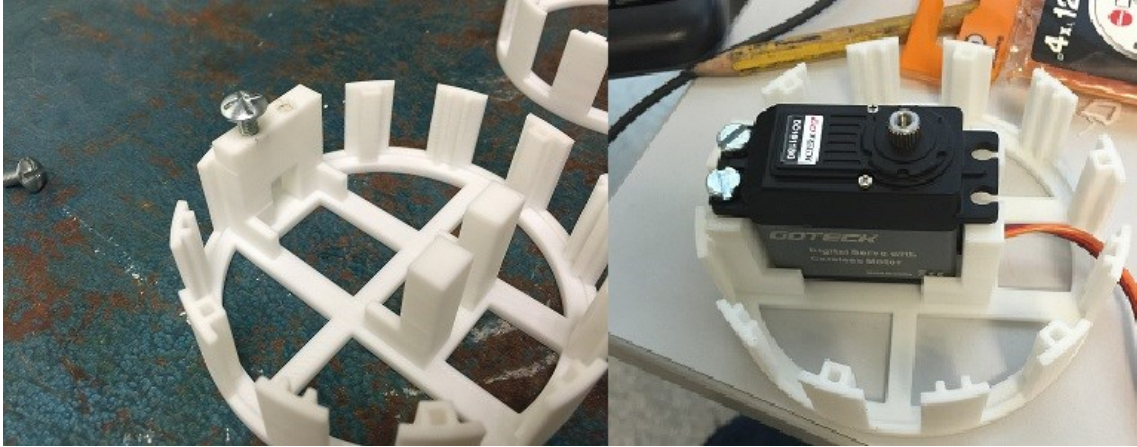


Figura 3.41 Fijado del servomotor de q_1

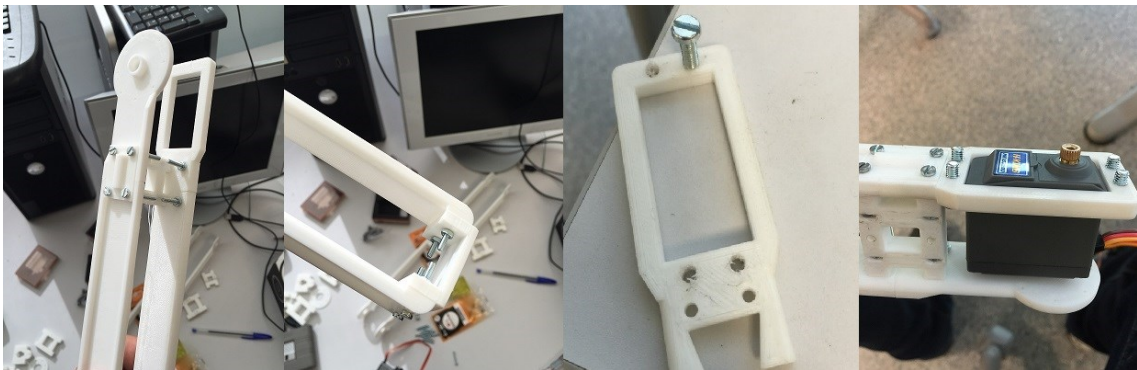


Figura 3.42 Preparación de la articulación q_3



Figura 3.43 Fijado del servomotor de la articulación q_2

Seguidamente, se realiza un primer intento para implementar las articulaciones. Aquí se hace necesaria ya la primera corrección, puesto que los servomotores adquiridos no cuentan con piezas circulares que anclar a su eje para transmitir el movimiento a las correspondientes arandelas, de forma que ha de idearse un nuevo mecanismo para este propósito.

La primera solución que se encuentra e implementa es combinar en una sola pieza impresa en PLA tanto la terminación con que el servomotor transmite el movimiento como la arandela con la que se une al eslabón correspondiente.

Se trata de 3 piezas muy similares que cuentan con un eje hueco donde se encaja con apriete el eje estriado del servomotor, de forma que, a priori, el eje del servomotor y la pieza giran solidariamente. Esta unión se refuerza mediante un tornillo que se introduce por un agujero en la parte superior de la pieza. Las 3 piezas solo se distinguen unas de otras por su espesor y por la longitud del eje hueco. Estas piezas se diseñan con ayuda de un técnico, utilizando el software de diseño NX de Siemens.

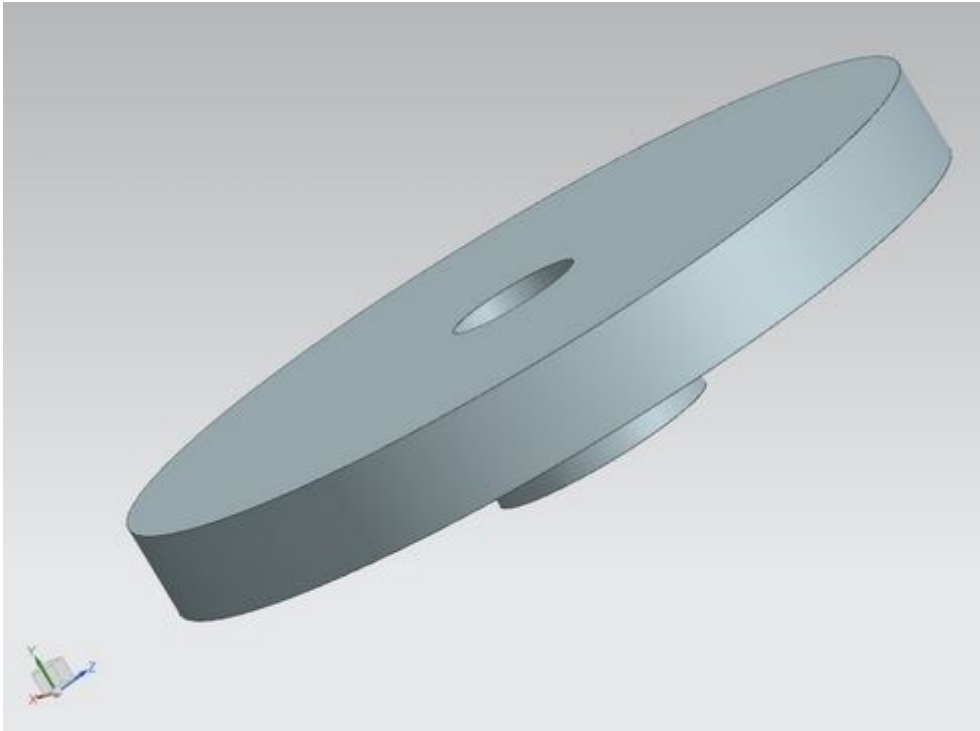


Figura 3.44 Diseño del primer intento de pieza para articulación

Tras ensamblar y montar las articulaciones, si bien consiguen transmitir inicialmente el movimiento, acaban por resultar incapaces de resistir el par transmitido por el servomotor, que lima los ejes huecos desde dentro y hace que las piezas resbalen, perdiendo así su capacidad de transmitir movimiento.

Diseño, construcción y control de un robot manipulador de 3 grados de libertad de bajo coste para el desarrollo de un manipulador móvil.

Se muestran a continuación imágenes (figuras 3.45, 3.46, 3.47) de dicho proceso de montaje y puesta a prueba:



Figura 3.45 Montaje de primera versión de articulación q_1



Figura 3.46 Preparación para prueba de la articulación q_3

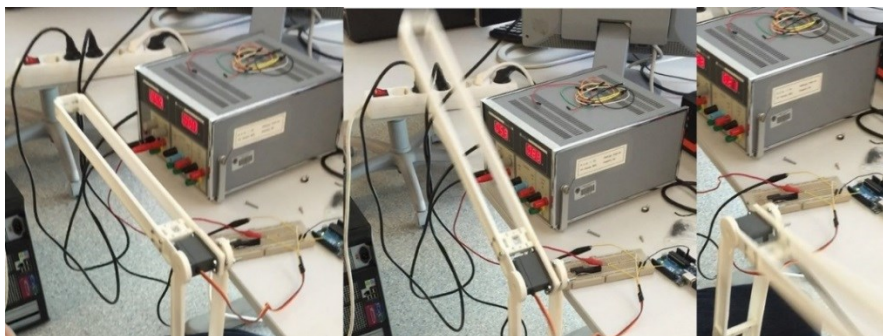


Figura 3.47 Prueba primer diseño para articulación q_2

Tras las pruebas, las piezas quedan muy desgastadas en el eje hueco, como se comprueba en la figura 3.48, donde ya no hay ni siquiera un leve estriado:



Figura 3.48 Detalle piezas de PLA para articulaciones desgastadas tras pruebas

Como conclusión de esta primera solución, para aplicaciones donde se vaya a transmitir un par elevado, se desaconseja utilizar materiales de impresión 3D al uso (PLA o ABS) como enlace directo entre el eje estriado del servomotor y el resto del robot.

La segunda solución que se encuentra, implementa y prueba con éxito es combinar piezas de impresión 3D con las piezas que el fabricante proporciona con cada servomotor, y que tienen su propio eje hueco ya estriado para encajar a la perfección con el del propio servomotor.

Con este fin, se diseñan 3 piezas cilíndricas que cuentan con un hueco para encajar las piezas del fabricante.

Como particularidades a señalar, para la articulación correspondiente a q_2 se requiere que el eje del servo pase a través de un agujero del eslabón 2, y un nervio existente en la pieza proporcionada por el fabricante del servomotor lo impide, a priori. Para que no sea necesario rediseñar una pieza se ha retirado el nervio utilizando una lima. En la figura 3.49 se puede comparar la pieza original con la pieza una vez retirado el nervio.



Figura 3.49 Detalle piezas servomotor antes y después de limado

Con esta fórmula sí se logra transmitir todo el par sin que se desgasten aceleradamente las articulaciones.

Así, tan solo hay que terminar de hacer los ajustes y correcciones necesarios, sobre todo a la hora de evitar que el anclaje de los servomotores interfiera con su correcto movimiento. Para ello se recurre a pequeñas modificaciones mediante operaciones como limado, taladrado para ocultar los cabezales de tornillo o recorte y limado de tornillos para conseguir una longitud ad hoc.



Figura 3.50 Distintas operaciones para evitar interferencia de tornillos en movimiento articular

Y, por último, se ensambla el brazo asegurando, como se ha indicado anteriormente, que la posición de los eslabones es coherente con el modelo cinemático previsto.

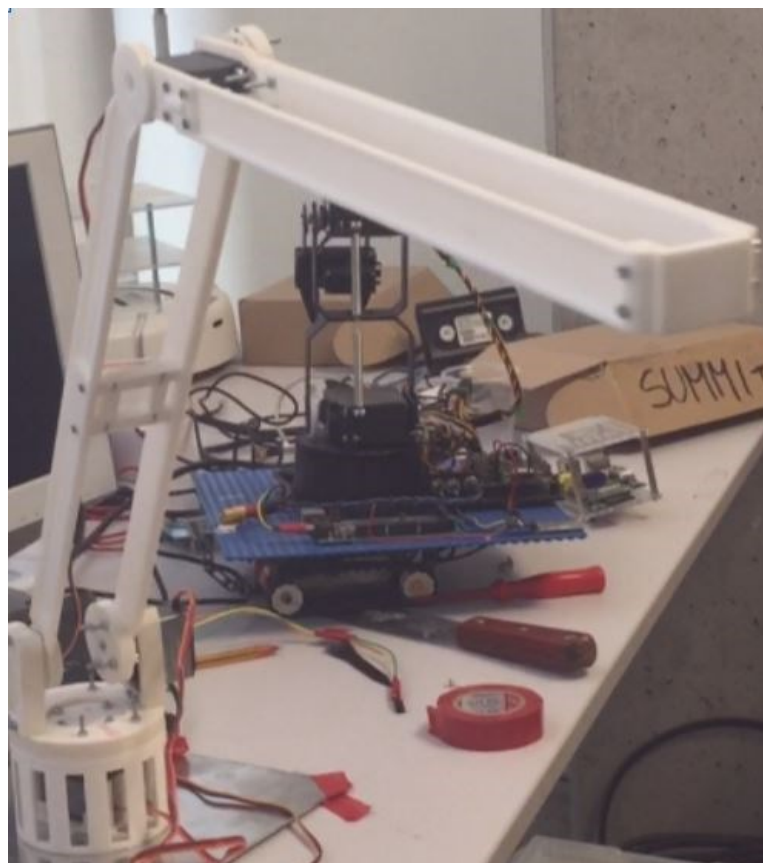


Figura 3.51 Brazo robótico, montaje final.

3.5- Arquitecturas finales:

3.5.1- PC y Arduino:

La placa Arduino se conecta mediante USB al PC, que tiene Ubuntu y ROS instalados, y los puertos PWM 9, 10 y 11 se conectan a los polos correspondientes a la recepción de señal de los servomotores q_1 , q_2 , q_3 .

Los polos de positivo y negativo de los servomotores se conectan a un empalme eléctrico soldado con estaño, que a su vez tiene dos puntos de conexión, uno positivo y otro neutro, para conectar a las salidas positivo y neutro de una fuente de alimentación externa. De este modo, se están conectando en paralelo los servomotores.

La fuente de alimentación externa es capaz de proporcionar más de 6V y más de 2.5A, que son los valores pico a los que llega el conjunto eléctrico. Además, el empalme eléctrico único se utiliza para evitar el uso de múltiples cables finos, empalmes y alargadores, que hacen que se pierda mucha energía eléctrica.

En concreto, para comprobar esta necesidad, se pone a prueba el brazo robótico comparando una instalación como la descrita con una que utiliza cables y empalmes de baja calidad, y se observa cómo en la instalación de baja calidad el brazo robótico no es capaz de obtener el suficiente par del servomotor de q_2 para hacer desplazamientos en sentido ascendente de más de 20-30°.

3.5.2- Raspberry Pi y Arduino:

Esta arquitectura tan solo difiere de la anterior en que se sustituye el PC por la Raspberry Pi con Raspbian y ROS instalados. En este caso tanto el montaje eléctrico como los nodos son iguales, se gana portabilidad pero se pierde rapidez y capacidad de procesamiento, siendo funcional en cualquier caso.

3.5.3- PC, Raspberry Pi y Arduino:

Esta arquitectura deriva de forma inmediata de las dos anteriores y consiste en conectar PC y Raspberry Pi en la misma red, para enviar la referencia a la Raspberry Pi desde el PC. Se libera al PC de gran parte de la carga de procesamiento y abre la puerta a aprovechar el procesador con ROS que tiene el Summit en futuros proyectos.

3.5.4- Raspberry Pi y módulo controlador de motores:

Se ha estudiado la sustitución de la placa Arduino por un módulo capaz de proporcionar potencia eléctrica y aplicar control por PWM, Gertbot, que está diseñado para motores de corriente continua y motores *stepper*.

Así, se deja una base para implementar un control que aproveche las ventajas de ROS y Raspberry Pi por un medio distinto al control sobre una placa Arduino y servomotores.

CAPÍTULO 4. CONCLUSIONES Y PROYECTOS FUTUROS:

4.1- Conclusiones:

Para cerrar la memoria descriptiva, se comprueba el cumplimiento de los objetivos marcados al inicio de la misma:

- Se ha satisfecho el objetivo principal del TFG, obteniendo como resultado final un brazo robótico de 3 grados de libertad con múltiples mecanismos de control cinemático que funcionan con distintas arquitecturas de control, y que cuenta con las dimensiones y modelo cinemático adecuadas para su uso sobre un *Summit*.
- Se han estudiado y resuelto con éxito tanto los problemas cinemáticos inverso y directo como distintos tipos de trayectoria articular.
- Se ha trabajado con ROS tanto en Ubuntu como en Raspbian, desarrollando y poniendo en funcionamiento *software*, además de haber logrado la comunicación con Arduino mediante *rosserial*, con detección de todas las particularidades y eventualidades que tiene este protocolo.
- Se ha realizado una documentación exhaustiva sobre servomotores, se han elegido adecuadamente y se han puesto a funcionar sin ningún inconveniente.
- Se han adquirido conocimientos tanto sobre las piezas que necesita un brazo robótico como sobre los aspectos más importantes a tener en cuenta a la hora de diseñarlas. En este sentido, se han obtenido piezas tanto modificadas a partir de otras preexistentes, como totalmente originales, sobre todo en cuanto a las articulaciones.
- Se ha experimentado y enumerado tanto el gran abanico de posibilidades y avances que ofrece la impresión 3D a la robótica, como los retos y limitaciones que plantea, fabricando por este medio todas las piezas del brazo, a excepción de los tornillos, las tuercas y los servomotores.
- Se ha dedicado un gran esfuerzo y tiempo para el montaje y la construcción del modelo, habiendo superado los distintos retos constructivos que se han ido planteando, de modo que se ha recurrido a distintas técnicas de mecanizado de piezas y al análisis real de la viabilidad de ciertas piezas que en el entorno virtual parecen perfectamente válidas.

Como se observa, tanto el objetivo fundamental como los objetivos secundarios que se desprendían de él han sido cumplidos con éxito.

4.2- Proyectos futuros:

A lo largo del trabajo ha habido dos cuestiones que se han destacado como de relevancia para futuros proyectos, porque sientan una base sólida de la que partir:

- 1- Se ha desarrollado un brazo robótico que efectivamente puede funcionar encima de la cubierta del *Summit* se encuentre este en la posición que se encuentre. Esto abre las puertas a realizar una integración a nivel de programación entre la computadora que utiliza el *Summit* y los paquetes con que trabaja el brazo

robótico, coordinando mediante ROS su funcionamiento, por ejemplo con modelos conjuntos de control con interfaz gráfica.

- 2- Se ha resuelto un problema de compatibilidad con ROS de un módulo para Raspberry Pi, muy versátil y de bajo coste, enfocado a motores. Ello es de interés para cualquier futura aplicación que pretenda aprovechar las ventajas que ofrece este módulo, pues no hay documentación abundante al respecto en internet.
- 3- El desarrollo de un prototipo como este de bajo coste con impresión 3D ha registrado no solo el modelo final, sino toda una serie de consideraciones que pueden ser de gran interés en aquellos ámbitos donde la extensión del uso de brazos robóticos de bajo coste pueda ser interesante, como por ejemplo la docencia, ya que el precio en este caso, tanto por el método de fabricación como por el *software* libre utilizado, puede ser muy competitivo.

CAPÍTULO 5. BIBLIOGRAFÍA:

5.1- Documentación:

- [1] Diccionario de la Real Academia Española. Definición de robot [en línea]: <http://dle.rae.es/?id=WYRlhzm>
- [2] Norma ISO 8373 [en línea]:
- [3] Material docente seminario introducción al control de robots, Laboratorio de Automatización y Control, GITI, UPV.
- [4] Proyecto educativo *Rover Ranch* asociado a la NASA. Tipos de robots: prime.jsc.nasa.gov/ROV/types.html
- [5] Antonio Barrientos, Luis Felipe Peñín y otros, 2007. Fundamentos de Robótica. Cinemática del robot.
- [6] Departamento de Arquitectura y Tecnología de Computadores, Universidad de Sevilla. Tema 4, parte 2. Coordenadas homogéneas y matriz de transformación homogénea [en línea]: http://icaro.eii.us.es/descargas/tema_4_parte_2.pdf
- [7] Mecapedia, Enciclopedia Virtual de Ingeniería Mecánica de la Universitat Jaume I. Par cinemático [en línea]: http://www.mecapedia.uji.es/par_cinematico.htm
- [8] Laboratorio de electrónica, UCLM. El servomotor [en línea]: <http://www.info-ab.uclm.es/labelec/solar/electronica/elementos/servomotor.htm>
- [9] Web de aeromodelismo *RC Helicopter Fun*. Servos [en línea]: <http://www.rchelicopterfun.com/rc-servos.html>
- [10] Corporación Futaba. Servos digitales [en línea]: www.futabarc.com/servos/digitalservos.pdf
- [11] Colaboradores de Wikipedia. Wikipedia. Motor eléctrico sin escobillas [en línea]: http://es.m.wikipedia.org/wiki/Motor_eléctrico_sin_escobillas
- [12] Enciclopedia virtual (Wiki) de ROS. Conceptos [en línea]: <http://wiki.ros.org/ROS/Concepts>
- [13] Enciclopedia virtual (Wiki) de ROS. Catkin [en línea]: <http://wiki.ros.org/catkin>
- [14] Enciclopedia virtual (Wiki) de ROS. CMakeLists.txt [en línea]: <http://wiki.ros.org/catkin/CMakeLists.txt>
- [15] Enciclopedia virtual (Wiki) de ROS. Package.xml [en línea]: <http://wiki.ros.org/catkin/package.xml>
- [16] Guía de Arduino [en línea]: <https://www.arduino.cc/en/Guide/Introduction>
- [17] Portal Linux. Instalar el IDE de Arduino en GNU/Linux [en línea]: <http://portallinux.es/instalar-el-ide-de-arduino-en-gnulinux/>
- [18] Enciclopedia virtual (Wiki) de ROS. Rosserial [en línea]: <http://wiki.ros.org/rosserial>

- [19] Enciclopedia virtual (Wiki) de ROS. Librerías de cliente de roserial [en línea]: http://wiki.ros.org/rosserial_client
- [20] Fundación Raspberry Pi. ¿Qué es Raspberry Pi? [en línea]: <http://www.raspberrypi.org/help/what-is-a-raspberry-pi/>
- [21] Web de Raspbian. Raspbian.Org [en línea]: <http://www.raspbian.org/>
- [22] Universidad de Santiago. Modelación Cinemática del Brazo Manipulador. Resolución del Problema Cinemático Inverso por Métodos Geométricos [en línea]: <http://www.udesantiagovirtual.cl/moodle2/mod/book/view.php?id=24918&chapterid=295>
- [23] Enciclopedia virtual (Wiki) de ROS. Tutoriales de *roserial_arduino*, “*Arduino IDE setup*” [en línea]: http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup
- [24] Enciclopedia virtual (Wiki) de ROS. Tutoriales de *roserial_arduino*, “*Servo controller*” [en línea]: http://wiki.ros.org/rosserial_arduino/Tutorials/Servo%20Controller
- [25] Enciclopedia virtual (Wiki) de ROS. Tutoriales de ROS. Instalar y configurar tu entorno ROS [en línea]: <http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment>
- [26] Enciclopedia virtual (Wiki) de ROS. Tutoriales de ROS. Crear un paquete [en línea]: <http://wiki.ros.org/ROS/Tutorials/CreatingPackage>
- [27] Enciclopedia virtual (Wiki) de ROS. Tutoriales de ROS. Construir un paquete [en línea]: <http://wiki.ros.org/ROS/Tutorials/BuildingPackages>
- [28] Enciclopedia virtual (Wiki) de ROS. Tutoriales de ROS. Crear un msg y srv de ROS [en línea]: <http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv>
- [29] Enciclopedia virtual (Wiki) de ROS. Tutoriales de ROS. Escribir un Publicador y Suscriptor simple (C++) [en línea]: <http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%29>
- [30] Enciclopedia virtual del espacio para desarrolladores sin ánimo de lucro *Jigsaw Renaissance*. ROS en RaspberryPi: http://wiki.jigsawrenaissance.org/ROS_on_RaspberryPi
- [31] Gertbot , página principal [en línea]: <http://www.gertbot.com>
- [32] Usuario *ancastro* de Thingiverse, brazo robótico de 6 gdl [en línea]: <http://www.thingiverse.com/thing:30163>
- [33] Base de datos de servos. Servo futaba s3003 [en línea]: <http://www.servodatabase.com/servo/futaba/s3003>
- [34] Base de datos de servos. Servo futaba s3004 [en línea]: <http://www.servodatabase.com/servo/futaba/s3004>
- [35] Colaboradores de Wikipedia. Wikipedia. Poliácido láctico [en línea]: http://es.m.wikipedia.org/wiki/Poliácido_láctico

[36] Colaboradores de Wikipedia. Wikipedia. Acrilonitrilo butadieno estireno [en línea]: http://es.m.wikipedia.org/wiki/Acrilonitrilo_butadieno_estireno

5.2- Imágenes:

[1] Tema 2, cinemática de robots, Robótica Industrial, MUII, ETSII.

[2] Buscar en imágenes de google “Sistema de referencia a derechas”:
<http://www.galia.fc.uaslp.mx/~medellin/Applets/Trans3D/transf7.gif>

[3] Modificada a partir de imagen encontrada en google buscando: “servo desmontado”
https://www.google.es/search?q=servo+desmontado&client=firefox-b-ab&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjqls3968rMAhXB7xQKHxsPCX0Q_AUIBygB&biw=1920&bih=947#imgrc=KnAIZCnuALwKRM%3A

[4] Modificada a partir de imagen encontrada en google buscando: “cable servo”
https://www.google.es/search?q=three+wire+plug+servo&client=firefox-b-ab&source=lnms&tbm=isch&sa=X&ved=0ahUKEwj9g-2O78rMAhVLbRQKHe7bDGcQ_AUIBygB&biw=1920&bih=947#tbm=isch&q=cable+servo&imgrc=Nh7deM9tpeTnM%3A

[5] Corporación Futaba. Servos digitales [en línea]:
www.futabarc.com/servos/digitalservos.pdf

[6] Enciclopedia virtual (Wiki) de ROS. Conceptos [en línea]:
<http://wiki.ros.org/ROS/Concepts>

[7] Imagen obtenida (con posibles modificaciones) de la fuente: Antonio Barrientos, Luis Felipe Peñín y otros, 2007. Fundamentos de Robótica. Cinemática del robot.

[8] Universidad de Santiago. Modelación Cinemática del Brazo Manipulador. Resolución del Problema Cinemático Inverso por Métodos Geométricos [en línea]:
<http://www.udesantiagoovirtual.cl/moodle2/mod/book/view.php?id=24918&chapterid=295>

PRESUPUESTO

1-JUSTIFICACIÓN DEL PRESUPUESTO:

Pese a su carácter derivado del aprendizaje académico, este trabajo fin de grado representa un proyecto material en que se han invertido unas horas de trabajo determinadas y se ha adquirido y utilizado material inventariable y fungible, de modo que es perfectamente reproducible como proyecto fuera del ámbito académico, y por tanto es crucial abordar la inversión que supondría, como en cualquier documento de proyecto de ingeniería.

Los datos del presupuesto en cuanto a retribuciones, así como los aspectos tratados en el estudio económico, se extraen del documento “Recomendaciones en la elaboración de presupuestos en actividades de I+D+I”, del Centro de Apoyo a la Innovación, la Investigación y la Transferencia de Tecnología (CTT) de la UPV del año 2015.

El carácter eminentemente de investigación de este trabajo ha decantado la elección de este documento del CTT como guía, en lugar de recurrir a otras formas de presupuesto.

2-ESTUDIO ECONÓMICO:

2.1-Costes de Personal:

El coste de personal se calcula mediante la fórmula:

$$\text{Coste de personal} = \text{Coste por hora} \times \text{Horas trabajadas}$$

El coste por hora corresponde al de un titulado medio contratado de forma temporal, de modo que se fija en 32 €/h, costes indirectos y de SS incluidos, que está entre el mínimo (26,8 €/h) y el máximo (36,2 €/h), tan solo ligeramente por encima del valor medio.

Tarea	Horas	Precio(€/h)	Coste parcial(€)
Formación y documentación	30	32,00	960
Instalación de software básico (ROS, Raspbian, IDE Arduino)	12		384
Cálculo modelo cinemático	8		256
Desarrollo de paquetes y nodos para ROS	80		2560
Instalación y estudio de roserial en PC y Raspberry	3		96
Desarrollo de programas para Arduino	2		64
Rediseño virtual del brazo robótico	15		480

Ensamblaje del brazo (incluidas las correcciones)	70		2240
Estudio comparativo de servomotores	5		160
Montaje eléctrico y pruebas	5		160
Implementación de los programas para Gertbot en Raspberry	10		320
Redacción de documentos	60		1920
TOTAL	300		9600

Sabiendo que el 32.1% corresponde a Seguridad Social, el 3.04% a indemnizaciones y 16.5 €/h se deben a costes indirectos, se puede realizar el siguiente desglose:

Concepto	Precio
Honorarios	1276.56
SS (32.1%)	3081.6
Indemnizaciones (3.04%)	291.84
Costes indirectos (16.5 €/h)	4950
TOTAL	9600

2.2-Material inventariable:

Como indica el CTT de la UPV, el coste de amortización del material inventariable se calcula como:

$$\text{Coste} = \text{Precio} \times \frac{\text{Meses de uso en el trabajo}}{12 \left(\frac{\text{meses}}{\text{año}} \right) \times \text{Periodo de amortización(años)}}$$

Concepto	Precio (€)	Meses de uso en el trabajo	Periodo de amortización (años)	Unidades	Coste (€)
Ordenador de sobremesa	599	4	6	1	33.28
Servomotor Goteck DC1611S	23.2	1	10	2	0.39
Servomotor Hobbyking HK-15328D	7.11	1	5	1	0.12
Arduino UNO	7.83	3	6	1	0.33

Raspberry Pi 1 modelo B	45.56	2	6	1	1.27
Monitor (para la Raspberry)	64.99	2	6	1	1.81
Ratón (para la Raspberry)	3.99	2	6	1	0.11
Teclado (para la Raspberry)	3.99	2	6	1	0.11
Cable ethernet (para la Raspberry)	7.49	2	6	1	0.21
Cable HDMI (para la Raspberry)	2.10	2	6	1	0.06
Tarjeta micro SD 16 GB y adaptador a SD (para la Raspberry)	4.45	2	6	1	0.12
Impresora 3D BCN3D sigma	2295	3	10	1	53.38
Taladro estacionario Rexon	135	2	10	1	2.25
Tornillo de banco	56.83	2	15	1	0.63
Sierra para metales	12.47	1	10	1	0.1
Juego de limas	11.54	2	15	1	0.13
Juego de destornilladores	11.89	2	15	1	0.13
Fuente de alimentación Grelco 30V 5A	462	2	10	1	7.7
Ubuntu 12.04, OpenScad, Arduino IDE, Raspbian jessie, ROS.	0	4	6	1	0
TOTAL					72.74

2.3-Material fungible:

Concepto	Coste (€)
Bobina PLA para impresión 3D (1ud)	20.83
Bolsa 30 tornillos 3x20mm(2uds)	3.46
Bolsa 30 tornillos 4x12mm(1ud)	1.73

Bolsa 30 tornillos 2.5x20mm(1ud)	1.73
Kit para resina epoxi (1ud)	8.95
Folios (200 uds)	5
Impresión y encuadernación	15
Corte de chapa de aluminio 20x20x2mm	1.5
TOTAL	58.2

3- RESUMEN DEL PRESUPUESTO:

Concepto	Coste (€)
Mano de obra (300h)	9600
Material inventariable	72.74
Material fungible	58.2
Subtotal	9730.94
IVA(21%)	2043.5
TOTAL	11774.44