

Trabajo Fin de Grado

Montaje y control coordinado de dos brazos
robóticos

Assembly and coordinated control of two robotics
arms

Autor

Josué David Poma Paqui

Director/es

Gonzalo López Nicolás
Rosario Aragües Muñoz

Montaje y control coordinado de dos brazos robóticos

RESUMEN

La aplicación de conceptos físicos y matemáticos sobre robots reales en el ámbito universitario puede verse impedida ya sea por temas de coste como por seguridad. Entonces se optará por robots de bajo coste económico. Se adquirirá dos brazos robóticos, PhantomX Pincher de 4 grados de libertad (GDL) para trabajar en un sistema multi-robot.

Tras su montaje, desarrollaremos el modelado geométrico de nuestro robot mediante matrices homogéneas y la parametrización de Denavit-Hartenberg, que junto a la cinemática inversa nos permitirán planificar trayectorias cartesianas.

La programación de los robots se realizará desde el entorno de MatLab. Nos apoyaremos en las librerías desarrolladas por el profesor australiano de robótica Peter Corke. No obstante, se detectarán errores durante la ejecución de estas. Se desarrollarán y se mostrarán las soluciones aplicadas.

Un sistema coordinado requiere poder leer continuamente la posición que marcan los encoders y además es importante plantearse cuán preciso es un robot para planificar la detección de finalización de un movimiento. Por ello, se graficará y cuantificará el error de posición.

Los motores no tienen un controlador de par para eliminar los errores de posición. Sin embargo, permiten modificar la curva de accionamiento de par. Mediante un buen ajuste se conseguirá un movimiento más suave, que no presente oscilaciones.

Se desarrollará un movimiento coordinado entre ambos robots, en el que se encargarán de manipular un sólido deformable, concretamente en girar 90° la suela de una zapatilla.

Contenido

0.	Introducción.....	1
0.1.	Motivación.....	1
0.2.	Objetivos.....	2
0.3.	Contenido.....	3
1.	Descripción del PhantomX Pincher.....	4
2.	Montaje del robot.....	6
2.1.	Herramientas necesarias.....	6
2.2.	Configuración de los motores.....	6
2.3.	Ensamblaje.....	7
2.3	Test de ensamblaje.....	8
3.	Modelado cinemático.....	9
3.1.	Entre el espacio cartesiano y el articular.....	9
3.2.	Método analítico algebraico.....	9
3.2.1	Localización espacial.....	9
3.2.2	Matrices de transformación homogénea.....	10
3.2.3	Modelo directo, Denavit-Hartenberg.....	11
3.2.4	Modelo inverso, resolución analítica algebraica.....	13
3.2.5	Análisis de las soluciones e implementación en MatLab.....	15
3.3.	Transformaciones entre radianes y marcas del motor.....	16
4.	Programación del robot.....	20
4.1	Especificaciones de los motores, aspectos importantes.....	20
4.2	Librerías.....	22
4.3	Conexión.....	23
4.4	Experimentos para evaluar la evolución de la posición y velocidad.....	25
4.5	Modificación de la librería Arbotix.m.....	28
4.5.1	Corrección y modificación de la función arb.setpos(id,pos,speed).....	28
4.5.2	Modificación de la función receive(arb).....	29
4.5.3	Adición de más funciones de lectura y escritura.....	32
4.6	Experimentos de ajuste del perfil de movimiento.....	32
5.	Aplicación.....	36
5.1	Objetivo.....	36
5.2	Planificación.....	37
5.3	Ejecución.....	41
5.4	Resultados y conclusiones.....	44
6	Conclusiones.....	45
7	Bibliografía.....	i

8	Índices.....	iv
8.2	Índice de figuras.....	iv
8.3	Índice de gráficas	v
8.4	Índice de tablas	v
A.	Anexos.....	1
A.1	Implementación del modelo geométrico en MatLab	1
A.2	Desarrollo detallado del método de variables intermedias	2
A.3	Programas y funciones de MatLab	8
	Phantom_X.m.....	8
	Matrices_de_paso.m.....	9
	Symdha.m (Función)	11
	Ikinе_PhantomX.m.....	12
	Inicializacion_comunicacion.m.....	15
	Arbotix.m (Versión modificada)	16
	Experimento_tipo1.m	40
	Obtencion_datos.m.....	42
	Experimento_tipo2.m	45
	Planificacion_recta.m	48
	Aplicación.m	51
A.4	Tabla de especificaciones del Dynamixel AX12-A.....	70
A.5	Circuito del motor Dynamixel AX12+	71

0. Introducción

0.1. Motivación

La robótica constituye uno de los pilares de la revolución industrial 4.0. Comenzando como novelas de ciencia ficción, se convierte en realidad por primera vez en 1961 con la aparición del UNIMATE [1], el primer robot industrial, en una planta de General Motors en New Jersey. Desde entonces, los robots industriales han evolucionado, pero también han sido causa de accidentes laborales. Por ello, durante muchos años se han encontrado en jaulas tanto a nivel industrial como a nivel universitario.

En los últimos años, se ha ido desarrollando una serie de robots con fuerzas y velocidades limitadas, y equipados con una serie de sensores que les permiten detenerse en caso de contacto o choque. Son los conocidos cobots, y que, en mayo de este año, las compañías tuvieron la oportunidad de ver los últimos avances y aplicaciones desarrolladas sobre éstos en el primer congreso de robótica colaborativa del mundo “WeAreCOBOTS” [2].

Ambas tecnologías presentan ventajas y desventajas, pero lo cierto es que son caras, lo cual complica mucho su uso práctico en las universidades. Por ello, la aplicación de los conceptos físicos y matemáticos se realizan frecuentemente sobre simulación en ordenador, en los que no podemos apreciar los problemas y limitaciones que pueden surgir en un robot real.

Por ello, el propósito principal de este trabajo es aplicar los conceptos abstractos aprendidos en el curso de robótica sobre el brazo robótico, PhantomX Pincher (ver Figura 1). Su bajo coste económico nos permite adquirir dos unidades, y trabajar en un sistema multi-robot. Su precio



Figura 1 PhantomX Pincher

está justificado por tener 4 grados de libertad, así como limitaciones en el control de posición, constituyéndose como un robot de hobby.

0.2. Objetivos

El objetivo principal es poner en marcha dos brazos robóticos PhantomX Pincher y desarrollar movimientos coordinados entre ambos robots. En la parte de coordinación se experimentará la manipulación de un sólido deformable, concretamente con una suela de calzado muy ligera de la empresa PrettyBallerinas. La idea de esto lo propusieron mis directores a raíz de la reciente colaboración de la Universidad de Zaragoza con empresas del sector del calzado, donde la automatización de procesos está muy por debajo de sectores como la automoción.

Los robots fueron adquiridos por el departamento a la compañía Trossen Robotics y sus manuales se encuentran en su página web. Para asegurar su correcto montaje, los leeremos detenidamente. Luego se verificará el ensamblaje desde la plataforma Arduino apoyándonos en las librerías realizadas por el fabricante [3].

Pero las tareas de coordinación, las haremos desde el entorno de MatLab que, a pesar de ser un programa de pago, nos ofrece una gran potencia para realizar los cálculos necesarios para la planificación de trayectorias. Además, nos permitirá simular el movimiento de ambos robots, graficar la evolución de la posición y velocidad de cada una de las articulaciones y realizar un movimiento coordinado. Para realizar órdenes desde MatLab sobre los robots nos apoyaremos en las librerías realizadas por el actual profesor australiano Peter Corke [4].

Por otro lado, aprovecharemos el software de MatLab para implementar el modelo geométrico que realizaremos de nuestro robot, así como la correcta transformación inversa que nos permitirá obtener las configuraciones articulares necesarias para poder realizar un movimiento cartesiano, como puede ser una trayectoria rectilínea.

La implementación de todo ello requiere un análisis y modelado previo sobre el PhantomX Pincher. Nos centraremos en la parte cinemática, porque en lo que se refiere a la parte dinámica, los motores Dynamixel AX-12A no incluyen un controlador de par, consecuencia de su bajo coste. Lo único que incluyen es un regulador de la curva de par (curva de tipo lineal) durante el accionamiento, pero no regula la magnitud del par a aplicar para lograr un error de posición nulo.

Por último, se realizará una evaluación y análisis experimental de los algoritmos desarrollados.

0.3. Contenido

Esta memoria está organizada como se indica a continuación:

El primer y segundo capítulo muestran una breve descripción de los robots y de los pasos a seguir para su montaje y verificación, así como los sitios webs donde se pueden consultar los manuales.

En el tercer capítulo se aborda en profundidad la cinemática. Se introducirá los diferentes tipos de localización que podemos usar, así como sus ventajas y desventajas. Se justificará el uso de matrices homogéneas, y se mostrará cómo se emplean para realizar un modelado geométrico del robot mediante los parámetros de Denavit-Hartenberg. Luego se mostrará las soluciones obtenidas en la cinemática inversa, así como un análisis de éstas. El desarrollo analítico de este método se incluye en el Anexo 2. El código de la implementación en MatLab se recoge también en los anexos, concretamente en el Anexo 3, que incluye los demás códigos que se desarrollan y/o modifican en este proyecto.

Al final de la parte cinemática se obtienen las funciones que nos permiten transformar radianes en marcas/pasos del motor y viceversa. Las marcas o pasos de motor son las unidades que debemos escribir sobre los registros de control.

En el cuarto capítulo, abordamos la programación de los robots desde el entorno de MatLab. Comenzamos introduciendo las especificaciones de los motores, así como los registros de control. La escritura y lectura de éstos se puede realizar con código a bajo o alto nivel; para realizarlo a alto nivel, se requieren librerías. Es aquí donde nos apoyaremos sobre las librerías del profesor Peter Corke. Y a continuación, se mostrará los pasos a seguir, para realizar una correcta conexión de todos los componentes implicados.

En ese mismo capítulo, se comentará las observaciones que se recogen en un primer tipo de experimentos sobre la evolución de la posición y la velocidad. También se explicará los problemas o dificultades, que encontramos a la hora de ejecutar las funciones que traían las librerías en nuestros primeros experimentos, y las correspondientes soluciones que desarrollamos. Después se realizará un segundo tipo de experimentos para verificar los cambios realizados y se evaluará la modificación de la des/aceleración angular.

En el quinto capítulo se planteará y se desarrollará el movimiento o aplicación que se llevará a cabo con la suela de calzado y que permitirá comprobar el correcto funcionamiento de ambos robots con todos los elementos analizados y desarrollados en los capítulos anteriores.

Y en sexto y último capítulo se mostrará las conclusiones generales acerca de este proyecto.

1. Descripción del PhantomX Pincher

En este trabajo se han usado dos brazos robóticos PhantomX Pincher. Este tipo de brazo robótico consta de 5 motores. Tal como se puede ver en la figura 2, los cuatro primeros motores forman articulaciones rotacionales, mientras que el último motor se encarga de abrir y cerrar la pinza, y no de variar la posición ni la orientación del efector final. Por este motivo, hablamos de que el robot tiene 4 grados de libertad (GDL) y no 5 GDL como afirman algunos autores de otras páginas webs.



Figura 2 Tipos de movimiento de cada una de las articulaciones del PhantomX Pincher [5]

Los robots se pueden adquirir a través de la web de Trossen Robotics [5], por un precio de 379,95\$/ud. (aproximadamente 345 €), que incluye además de las piezas del robot (ver

Figura 3), una placa ArbotiX-M, un adaptador de AC a DC¹ y un cable FTDI 5.0 V (ver Figura 4) y 5 motores paso a paso Dynamixel AX-12A (ver Figura 5). Se trata por tanto de un robot muy económico que nos permitirá acercarnos a la robótica industrial. Además, su plataforma viene diseñada para que pueda ser montado sobre un robot móvil TurtleBot (ver Figura 6).



Figura 3 Partes mecánicas del robot

¹ Necesario un adaptador de enchufe americano a europeo.



Figura 4 Placa ArbotiX-M, adaptador AC-DC y cable FTDI 5.0



Figura 5 Motores Dynamixel AX-12A

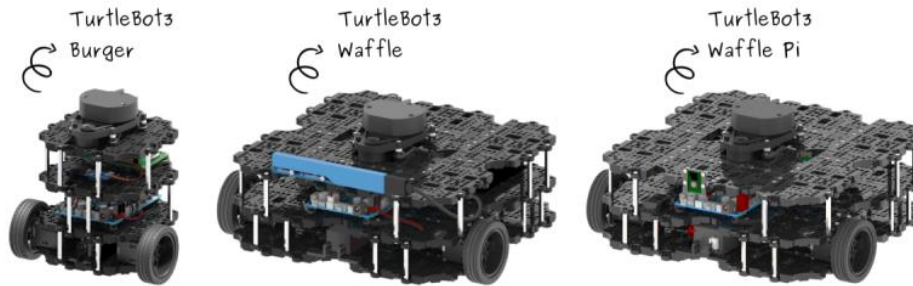
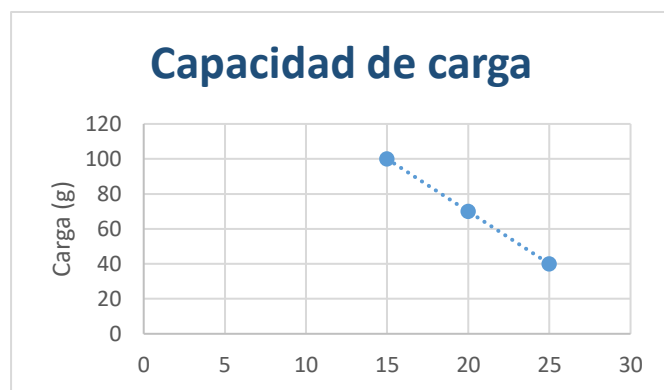


Figura 6 Distintas plataformas TurtleBots [37]

Nuestro brazo robótico es ligero, pesa tan sólo 550 gramos. En el eje horizontal el brazo alcanza 31 cm, mientras que en el vertical 35 cm. La fuerza de agarre de la muñeca es de 0.5 kg y en cuanto al peso que es capaz de levantar, según recomendaciones del fabricante son las que se muestran en la Gráfica 1.

distancia (cm)	carga (g)
15	100
20	70
25	40

Tabla 1 Capacidad de carga con respecto a la distancia horizontal



Gráfica 1 Capacidad de carga con respecto a la distancia horizontal

2. Montaje del robot

El robot se entrega desmontado y el primer paso, antes de montar, es adquirir las herramientas necesarias para su correcto montaje. También, previamente al montaje, se descargará el programa DynaManager [6] con el que configuraremos los motores. Después se realizará el montaje y finalmente para poder comprobar su funcionamiento se instalará Arduino, los FTDI drivers y las librerías que incluyen un programa de testing.

2.1. Herramientas necesarias

Antes de realizar el montaje es necesario adquirir:

- Tijeras.
- Lija de grano fino.
- Alicates pequeño (Ver Figura 7).
- Pegamento instantáneo (Ver figura 7).
- Juego de destornilladores (recomendado)², que incluya

(ver Figura 7 y 8):

- Llave hexagonal de 1.5 mm.
- Llave hexagonal de 2.5 mm.
- Llave Philips (estrella) PH-2.



Figura 7 Kit de herramientas



Figura 8 Llaves necesarias

2.2. Configuración de los motores

La configuración de los motores previo al montaje es muy importante. En este paso debemos asignarle una identificación (ID) a través de software (DynaManager [6]) así como una etiqueta, para tenerlo en cuenta a la hora de montar el robot. Con ese mismo programa o manualmente debemos “centrar” los motores; es decir, asegurarnos de que las hendiduras que señalamos con las flechas azules en la figura 9, queden totalmente alineadas. Esto se realiza con el objetivo de que el servo quede situado a la mitad de su rango de funcionamiento (0°-300°).



Figura 9 Motor: posición inicial incorrecta

² Las herramientas que vienen en el pack son de una calidad baja.

2.3. Ensamblaje.

Una vez configurado los motores, podemos comenzar a ensamblar las piezas. Para ello, seguiremos las instrucciones que nos marca Trossen Robotics para el montaje de la pinza [7] y del brazo [8]. En la figura 10 se puede observar las fotos del montaje:

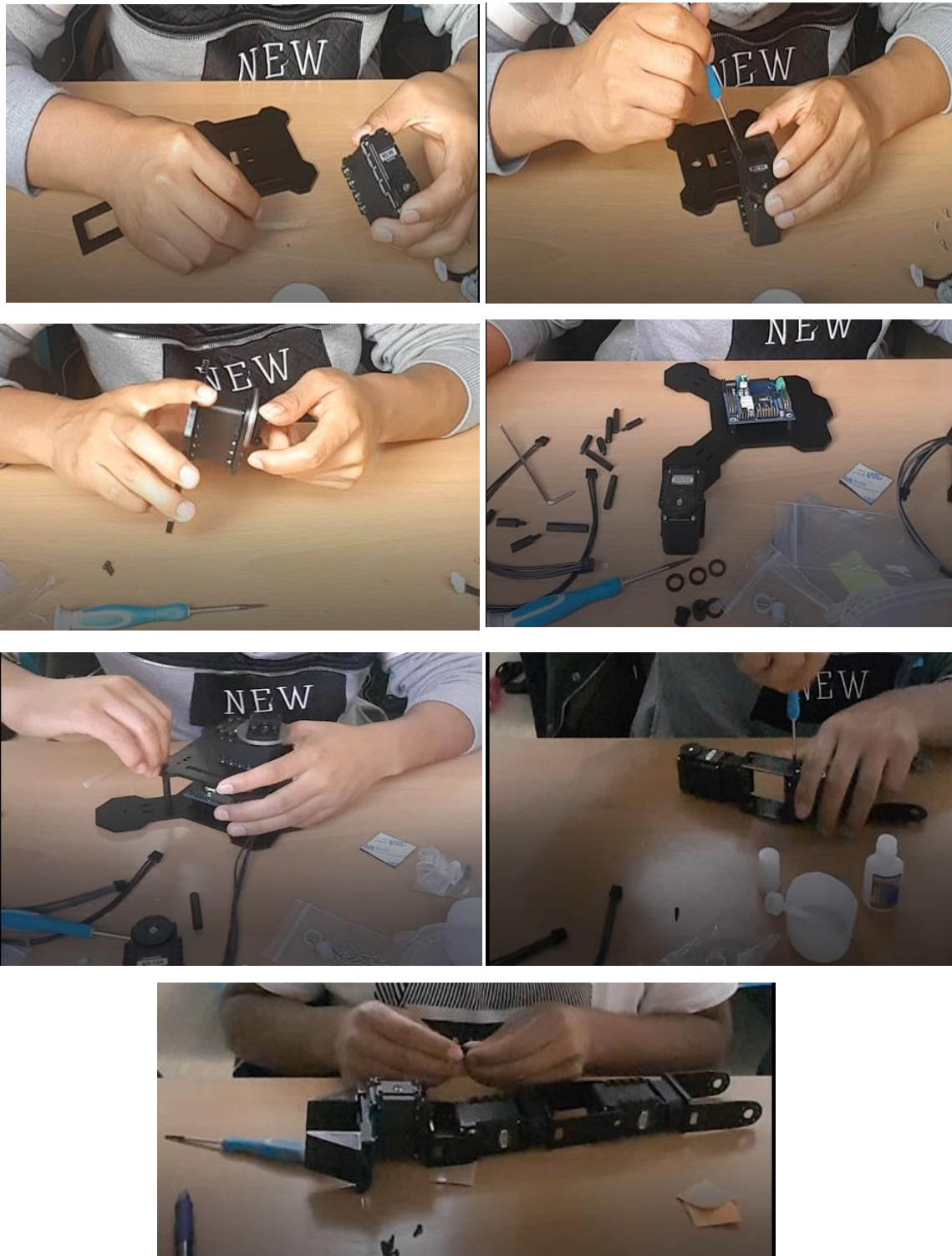


Figura 10 Fotos del montaje del robot

Cuando se inserten las tuercas en los motores, hay que asegurarse que las tuercas encajen perfectamente, de lo contrario, dará problemas a la hora de atornillar. Como recomendación, usar las tuercas que vienen en las cajas de los motores, en vez de las tuercas que vienen en el kit del robot, porque de estas últimas, hay algunas que no encajan por ser más grandes, debido a tolerancias de fabricación.

Dado que al atornillar se usa pegamento, hay que asegurarse que se está realizando correctamente el montaje.

Cuando montemos la pinza, se observará que las garras disponen de un pequeño agujero en comparación al tornillo que hay que insertar. A pesar de las líneas de flujo que se observan en las garras debido a su proceso de conformado, debemos agrandar el agujero atornillando únicamente el tornillo, luego desatornillaremos y realizamos el montaje.

2.3 Test de ensamblaje.

Al finalizar el montaje, procederemos a comprobar su funcionamiento. Para ello, comenzamos con la descarga de Arduino v. 1.0.6 [9]. Luego instalaremos los FTDI drivers [3] y las librerías que proporciona el fabricante [3]. Dentro de ellas se encuentra el programa, PincherTest.ino, que nos permitirá verificar su ensamblaje.

Todos los pasos para seguir, así como el vídeo en el que se muestra el movimiento que debe realizar el robot se encuentra en la web [10]. En dicha prueba, se realiza secuencialmente el movimiento individual de cada una de las articulaciones, tal como se puede apreciar en la figura 11. De esta manera, al completar todos estos pasos nos habremos asegurado de que el montaje e instalación de software ha sido el correcto.



Figura 11 Fotos de la prueba realizada para comprobar su correcto montaje. Movimiento de la segunda, tercera y cuarta articulación (de izquierda a derecha).

3. Modelado cinemático

Ahora que ya hemos montado el robot y comprobado su correcto montaje, vamos a abordar su cinemática. Para ello, partiremos explicando brevemente lo importante que es tener una función que nos permita pasar de trayectorias cartesianas a valores articulares, que serán las consignas. Luego se explicarán las herramientas matemáticas necesarias y las aplicaremos para nuestro robot. Finalmente obtendremos dicha función y la implementaremos en MatLab.

3.1. Entre el espacio cartesiano y el articular.

Nuestro manipulador consta de 4 GDL, o lo que es lo mismo, cuatro variables articulares, θ_i . En función del valor que tengan, permitirá que la pinza alcance una posición u otra, así como una orientación distinta, pero, teniendo en cuenta que no podremos alcanzar una localización arbitraria debido a que sólo tenemos 4 GDL.

El control se realiza en el espacio articular, pero muchas tareas requieren movimientos cartesianos, como moverse en línea recta o describir una circunferencia. Debemos, por tanto, ser capaces de realizar una transformación desde el espacio cartesiano, $\mathbf{X} = (x,y,z,\varphi)^T$, al espacio articular $\mathbf{Q} = (\theta_1, \theta_2, \theta_3, \theta_4)$, conocida como transformación inversa. Existen métodos numéricos y analíticos. Estos últimos han tomado gran importancia, porque muchos procesos industriales requieren transformaciones eficientes en tiempo real [11].

Dentro de los métodos analíticos, hay dos enfoques, uno geométrico y otro algebraico. El primero de ellos, resulta más sencillo cuando tenemos pocos grados de libertad, pero no es sistemático, a diferencia del último. En cualquier caso, hay que tener en cuenta la multiplicidad de soluciones que se pueda presentar, así como posibles singularidades.

3.2. Método analítico algebraico.

3.2.1 Localización espacial.

Antes de adentrarnos en el método, debemos caracterizar la posición y orientación del efector final con respecto a la referencia base. Lo podemos hacer separadamente, entonces hablaremos de vector de localización, o conjuntamente, usando lo que se conoce como matriz de transformación homogénea.

El vector de localización está compuesto por un vector de posición y otro de orientación. En el caso de la posición se puede usar el sistema cartesiano, polar o cilíndrico. Para la orientación, se puede usar los ángulos de Euler, par de rotación y cuaternios. El inconveniente que presenta esta caracterización es que complica la composición de rotaciones [12] en el caso de ángulos de Euler y par de rotación. En cambio, los cuaternios, a pesar de permitir una composición más fácil de rotaciones, evitar singularidades [13] y ser más compactos que las matrices homogéneas, en las que necesitamos 12 elementos, no nos permiten representar traslaciones por sí solos [12], [14]. Además, usar este tipo de caracterización, no nos permite transformar un vector en otro; es decir, al tratar por separado posición y orientación no podemos tener una “función” o “transformación” a la que aportamos como entrada un vector de localización inicial, $\mathbf{X}_i = (x,y,z,\varphi,\theta,\psi)^T$, y obtenemos un vector de localización final, $\mathbf{X}_f = (x,y,z,\varphi,\theta,\psi)^T$, al que hemos aplicado una traslación en cualquier dirección y/o una rotación en torno a cualquier eje.

3.2.2 Matrices de transformación homogénea

Es por ello, que usaremos la matriz de transformación homogénea. A ella se le asocia tres conceptos o significados [15], [16], por un lado, permite posicionar y orientar el elemento terminal, pues dicha matriz, contiene el vector de posición, \vec{p} , la orientación expresada por vectores ortonormales, $\vec{n}, \vec{o}, \vec{a}$, que constituyen una base asociada al elemento terminal (ver figura 12). Todos ellos expresados en la referencia fija.

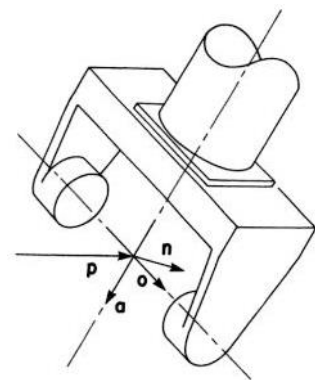


Figura 12 Descripción gráfica de localización (posición + orientación) [38]

$$T = \begin{pmatrix} \vec{n} & \vec{o} & \vec{a} & \vec{p} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \text{orientación} & \text{posición} \\ \text{perspectiva} & \text{escala} \end{pmatrix}$$

Por otro lado, nos permiten transformar localizaciones. Dada una localización expresada en matriz de transformación homogénea, podemos aplicarle una rotación, R_{3x3} , seguida de una traslación, \vec{p} , y obtener una nueva localización solamente premultiplicando.

$$T = \begin{pmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & s_{1x1} \end{pmatrix} = \begin{pmatrix} \text{rotación} & \text{traslación} \\ \text{perspectiva} & \text{escala} \end{pmatrix}$$

El uso de estas matrices está muy extendido en otros campos como el tratamiento de imágenes, es por ello, que presenta un vector de perspectiva, f_{1x3} , y un escalar que es el factor de escala.

En el campo de la robótica, no se emplean, por lo que el vector de perspectiva es el vector nulo, y el factor de escala es la unidad.

Por último, estas matrices nos permiten realizar cambios de base.

3.2.3 Modelo directo, Denavit-Hartenberg

A continuación, describiremos la matriz de transformación homogénea entre la referencia fija y el efector final, 0T_n , como un producto de n transformaciones homogéneas, una por cada articulación.

$${}^0T_n = {}^0T_1 \cdot {}^1T_2 \cdots {}^{n-1}T_n$$

El método propuesto por Denavit-Hartenberg asigna a cada eslabón un sistema de referencia bajo ciertas restricciones [17], de forma que, sólo necesitaremos 4 movimientos (2 de traslación y 2 de rotación) en vez de 6 (3 de rotación y 3 de traslación) para pasar de una base a otra.

La matriz de transformación, ${}^{i-1}T_i$, asociada al eslabón i [14] [18], es función de 4 parámetros (θ, d, a, α) que dependen de la geometría y posición relativa de los eslabones. En función del tipo de articulación, rotatoria o prismática, el parámetro que será una variable para controlar es la θ o d , respectivamente. En nuestro caso, todas las articulaciones son rotacionales.

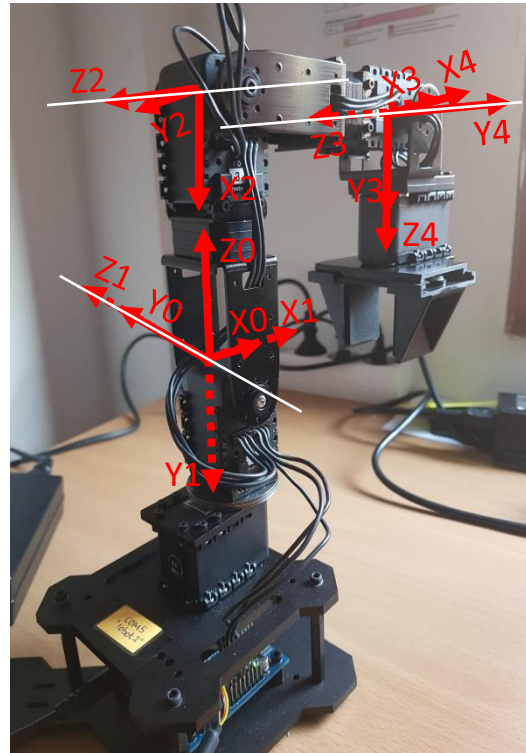


Figura 13 Sistemas de referencias según DH sobre el robot en reposo

$${}^{i-1}T_i = Rot(z, \theta_i) \cdot Trasl(z, d_i) \cdot Trasl(x, a_i) \cdot Rot(x, \alpha_i)$$

Aplicando el método de Denavit-Hartenberg erg hemos obtenido los parámetros³ (ver Tabla 2) de cada una de las matrices homogéneas para la posición de reposo que hemos considerado (ver Figura 13). Como se puede ver en la tabla 2, nuestra posición angular de reposo, θ_i , no coincide con el valor nulo de las variables articulares 2 y 3. Por lo que será necesario definir un offset.

	d_i	a_i	α_i	θ_i
1	0	0	-90	0
2	0	-0,107	0	90
3	0	0,107	0	-90
4	0	0	-90	0

Tabla 2 Parámetros DH de nuestro robot en reposo.

Además, la última base, $\vec{x}_4, \vec{y}_4, \vec{z}_4$, no posiciona el extremo de la pinza (ver Figura 13), por lo que añadimos una transformación homogénea adicional que contiene una traslación en la dirección \vec{z}_4 de 109 [mm].

$${}^4T_{pinza} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.109 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

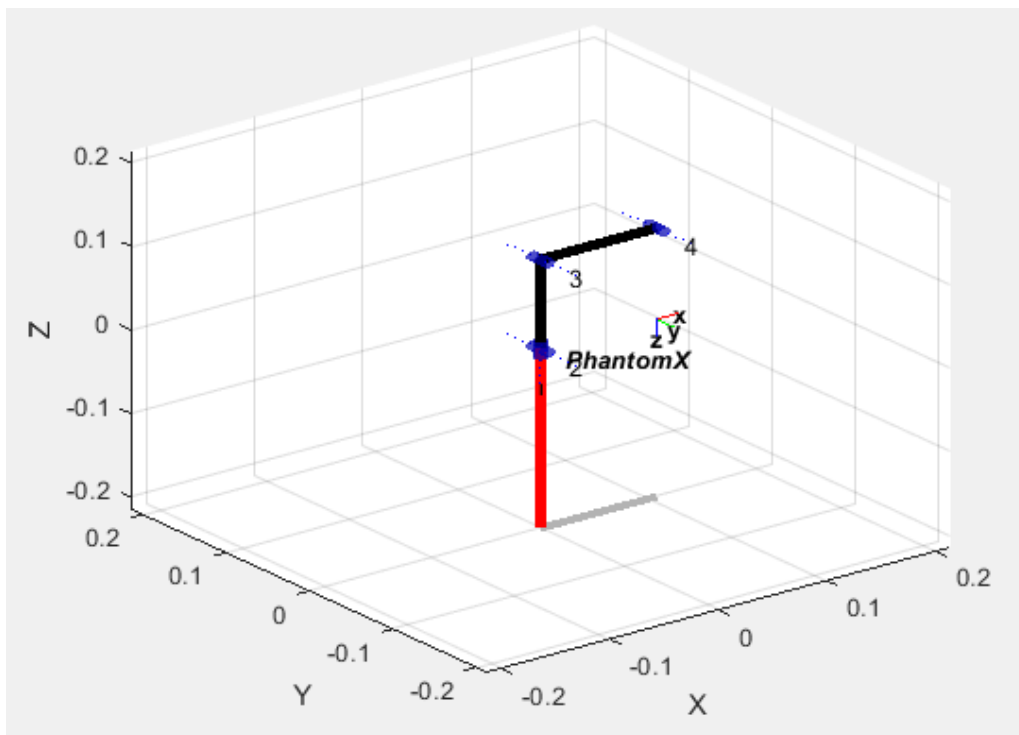


Figura 14 Modelado en MatLab del robot, en reposo.

A continuación, introduciremos los parámetros de DH, el offset y ${}^4T_{pinza}$ en el software de MatLab (ver Anexo 1) con el objetivo de comprobar que el cálculo de los parámetros se ha

³ Las longitudes se expresan en metros y los ángulos en grados sexagesimales

realizado correctamente. Si graficamos el modelo (ver Figura 14) vemos que coincide con la posición de reposo planteada (ver Figura 13).

La implementación se ha realizado con la ayuda de las funciones que trae la *Robotics Toolbox* desarrollada por el profesor Peter Corke. Las *Toolboxes* son complementos que traen funciones destinadas a ciertos campos de estudio. Éstos pueden ser desarrollados por MathWorks, pero también por terceros. En nuestro caso hemos elegido trabajar con la *Robotics Toolbox* de Corke porque se usó en el curso de robótica y es software libre.

3.2.4 Modelo inverso, resolución analítica algebraica

Es el momento de centrarnos en la resolución del problema inverso; es decir, obtener las coordenadas articulares dadas una posición y orientación del destino u objeto a alcanzar.

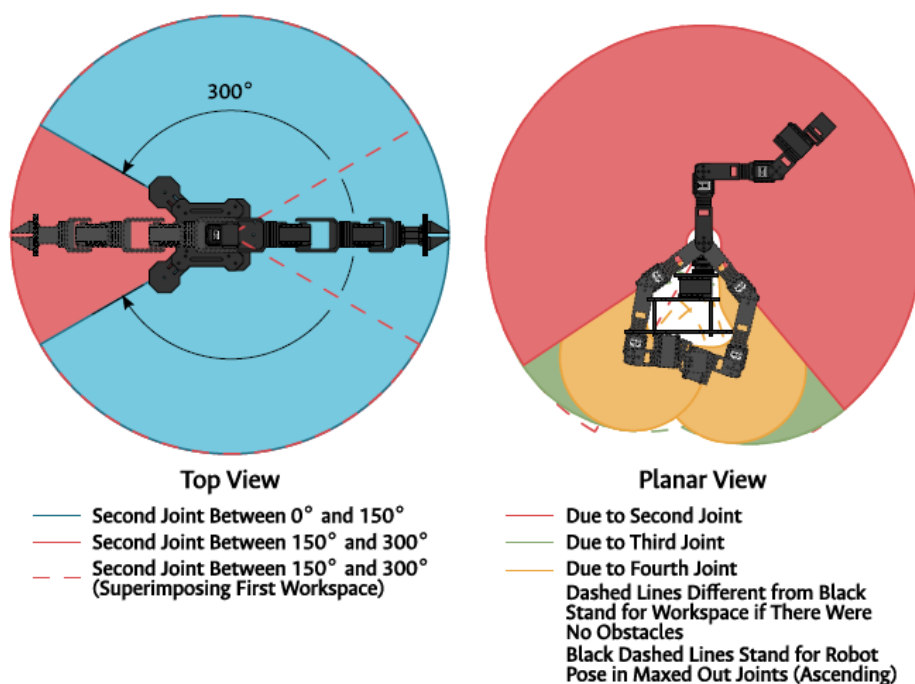


Figura 15 Espacio de trabajo ampliado [39]

De antemano, debemos ser conscientes de las limitaciones que tiene nuestro robot en cuanto a tareas que requieran 5 o 6 GDL no tendrán solución en el espacio articular debido a que contamos con 4 GDL.

Por otro lado, las limitaciones articulares junto con la geometría del soporte del robot delimitan el número de soluciones y definen nuestro espacio de trabajo (ver Figura 15). Cualquier localización fuera de esta zona, no tiene solución.

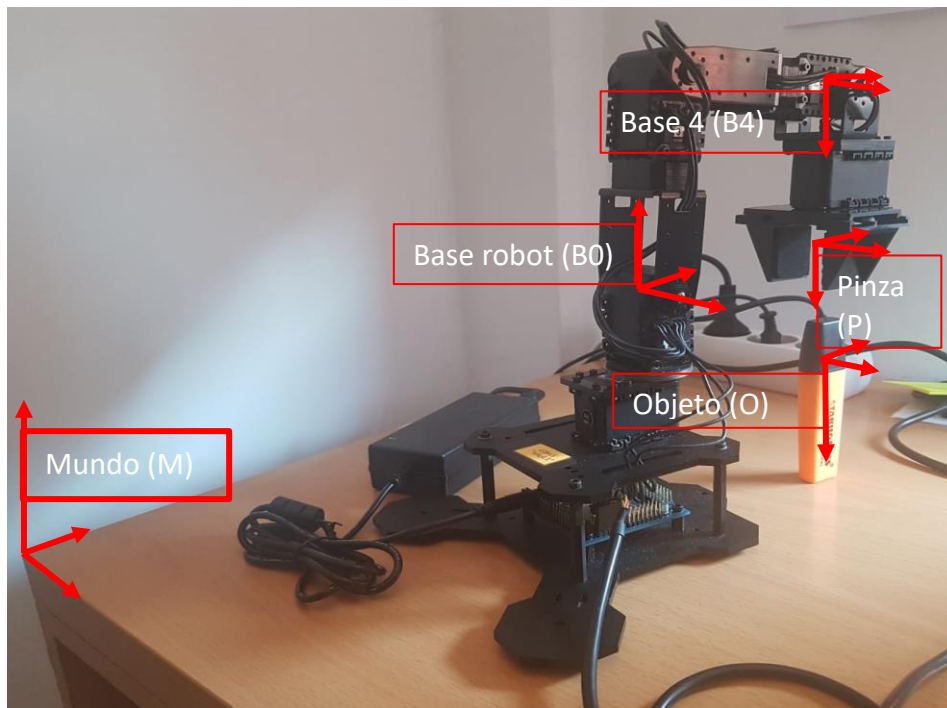


Figura 16 Esquema de las referencias sobre las que trabajaremos

Por lo general, la posición y orientación del objeto serán dadas sobre una referencia mundo, ${}^M T_O$ (ver Figura 16). Pero a nosotros nos interesa tener como valores numéricos de referencia los elementos de la matriz ${}^{B0} T_{B4}$. Así pues, realizamos un par de transformaciones:

$${}^{B0} T_{B4} = ({}^M T_{B0})^{-1} \cdot {}^M T_O \cdot ({}^{B4} T_P)^{-1}$$

Sustituyendo por variables simbólicas:

$${}^{B0} T_{B4} = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

E igualando a la matriz de transformación, ${}^0 T_4 = f(\theta_1, \theta_2, \theta_3, \theta_4)$, obtenemos un sistema de 12 ecuaciones no lineales y dependientes con 4 incógnitas, lo que dificulta la resolución. Por lo que se sigue el método heurístico de variables intermedias [17]. El cálculo detallado se encuentra en el Anexo 3.

A continuación, se muestra las soluciones alcanzadas, suponiendo que no hay límites articulares.

$$\theta_1 = \text{atan2}(o_x, -o_y)$$

$$\theta_2 = \text{atan2}(V_{124}, V_{114}) \pm \arccos\left(\frac{\gamma}{+\sqrt{V_{114}^2 + V_{124}^2}}\right)$$

$$\theta_3 = \text{atan2}\left(\frac{V_{224}}{a_3}, \frac{V_{214}}{a_3}\right)$$

$$\theta_4 = \text{atan2}(V_{321}, V_{311})$$

Donde:

$$\gamma = \frac{a_3^2 - a_2^2 - V_{114}^2 - V_{124}^2}{-2a_2}$$

$$V_{114} = p_x \cos(\theta_1) + p_y \sin(\theta_1)$$

$$V_{124} = -p_z$$

$$V_{224} = -p_z \cos(\theta_2) - (p_x \cos(\theta_1) + p_y \sin(\theta_1)) \sin(\theta_2)$$

$$V_{214} = (p_x \cos(\theta_1) + p_y \sin(\theta_1)) \cos(\theta_2) - a_2 + (-p_z) \sin(\theta_2)$$

$$V_{321} = (V_{121}C_2 - V_{111}S_2)C_3 - (V_{111}C_2 + V_{121}S_2)S_3$$

$$V_{311} = (V_{111}C_2 + V_{121}S_2)C_3 + (V_{121}C_2 - V_{111}S_2)S_3$$

Donde:

$$V_{121} = -n_z$$

$$V_{111} = n_x \cos(\theta_1) + n_y \sin(\theta_1)$$

3.2.5 Análisis de las soluciones e implementación en MatLab

Como se puede observar sólo para el ángulo θ_2 , tenemos una doble solución. Conceptualmente se traduce a que el robot puede alcanzar una localización con una configuración codo arriba y otra con el codo abajo.

Sin embargo, θ_2 no tiene solución si:

$$V_{124}^2 + V_{114}^2 < \gamma^2$$

Cuya inecuación se cumplirá si la localización se encuentra fuera del área de trabajo.

Ningún otro ángulo presenta doble solución, por lo que matemáticamente el número máximo de soluciones para una configuración puede ser de dos. En muchas ocasiones se reduce a una única solución o incluso a ninguna, debido a límites articulares o a orientaciones de la pinza inalcanzables, debido a que sólo disponemos de 4 GDL.

No se ha realizado ningún cálculo para el valor de θ_5 , porque el último motor sólo permite abrir y cerrar pinza; es decir, no es un GDL de nuestro robot. Su valor, se hallará experimentalmente, en función del objeto que queramos manipular.

Las ecuaciones de la resolución analítica se han implementado finalmente en la función ***ikinePhantomX.m*** (ver Anexo 3). La estructura de la función está basada en la ***ikine560.m***⁴, versión adaptada a la de Robert Biro [19] por Josechu Guerrero para el curso de robótica de la Universidad de Zaragoza.

3.3. Transformaciones entre radianes y marcas del motor

En el apartado anterior, al implementar las funciones en MatLab, las posiciones angulares de cada articulación quedaban expresadas en radianes. Los motores de los brazos robóticos en cambio trabajan con otro tipo de unidades, denominadas marcas o pasos y su uso se restringe a un modo de funcionamiento de los motores.

Nuestros motores, tienen dos modos de funcionamiento conmutables [20]. El modo *Wheel* hace que el motor gire en el estacionario a la velocidad deseada, lo cual es útil, por ejemplo, para mover un vehículo.

El otro modo, *Joint*, tiene un rango limitado de movimiento (ver Figura 17) de hasta 300°. Este rango angular se encuentra dividido en 1024 pasos o marcas (ver Figura 17); es decir, la posición angular se encuentra discretizada. Con este tipo de discretización se tiene una resolución de 0.29°. La vista de la figura 18 corresponde con la vista frontal del motor Dynamixel (ver Figura 18).

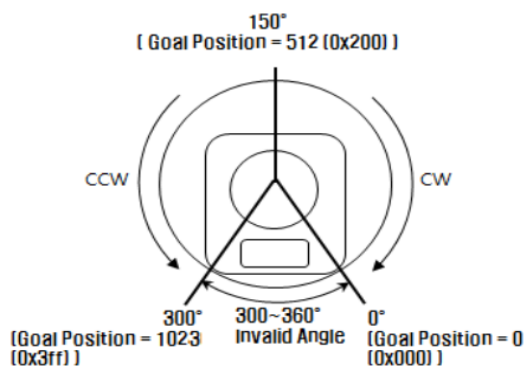


Figura 17 Límites articulares, Joint mode [21]



Figura 18 Vista frontal del motor

⁴ Corresponde a la cinemática inversa del PUMA 560

Este último modo es el que viene configurado de fábrica y con el que trabajaremos. Pero, debemos prestar especial cuidado, porque ciertas configuraciones que están dentro del rango de movimiento pueden ocasionar que el robot colisione contra sí mismo. Para evitar este problema de forma rápida, los primeros puntos de partida de las trayectorias de nuestros robots serán obtenidas por guiado manual, luego se planificará la trayectoria y se simulará el movimiento.

A continuación, vamos a ver qué marcas corresponden a la posición de reposo (ver Figura 13). Para poder relacionarlo tendremos en cuenta también las figuras 17 y 18. Entonces se observará que la posición inicial de los motores expresado en marcas no es el mismo para todos los motores. La posición inicial en marcas de los motores 1, 2 y 5 es de 512 (ver Figura 19), mientras que para los motores 3 y 4 es de 814 (ver Figura 20).

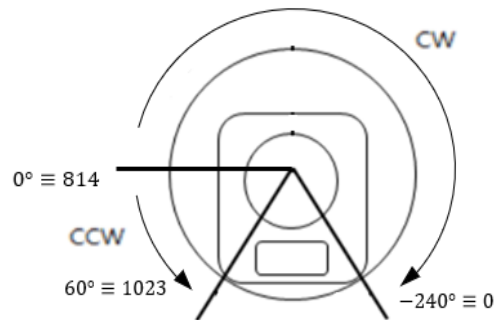
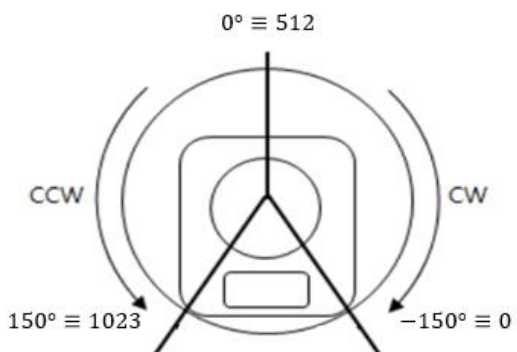
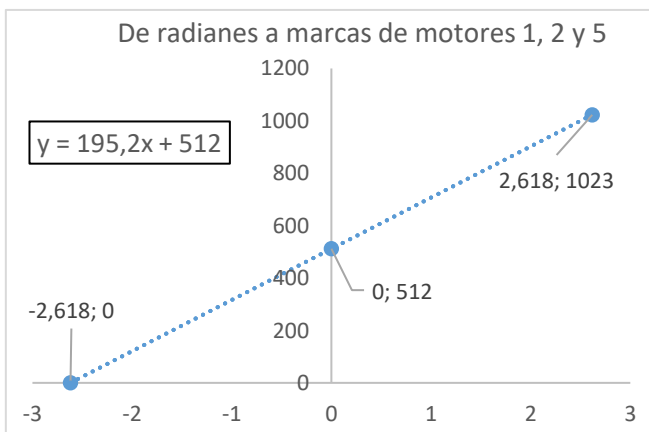


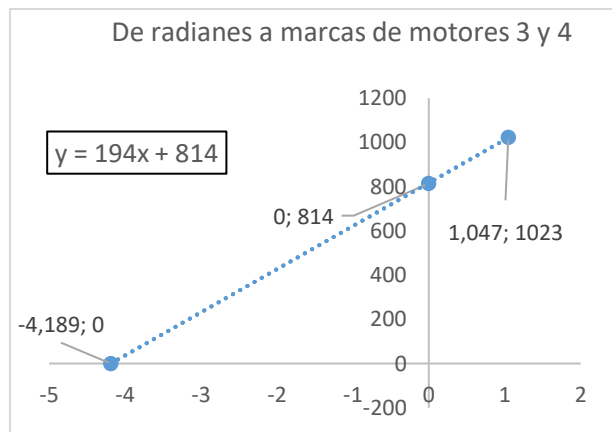
Figura 19 Posición inicial (reposo) de los motores 1,2 y 5

Figura 20 Posición inicial (reposo) de los motores 3 y 4.

La diferencia entre unos motores y otros nos lleva a plantear dos funciones distintas para cada uno, en las que se obtendrá los ángulos expresados en marcas en función de las soluciones obtenidas por la cinemática inversa que están expresadas en radianes. Las dos funciones consideradas serán de tipo lineal. Para los motores 1, 2 y 5 obtenemos la función que se observa en la gráfica 3, mientras que para los motores 3 y 4 se obtiene la que se encuentra en la gráfica 2.

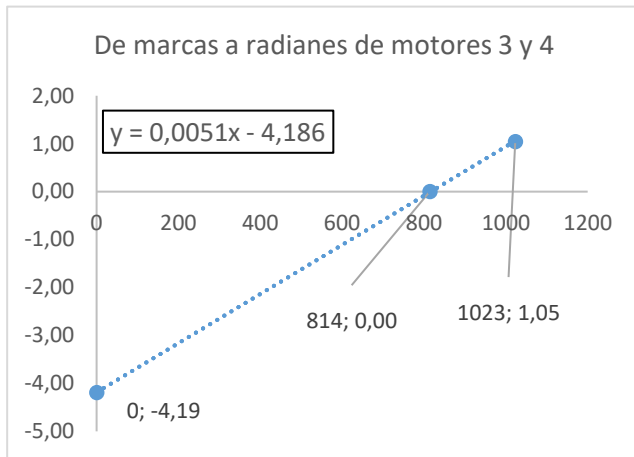


Gráfica 3 De radianes a marcas de motores 3 y 4

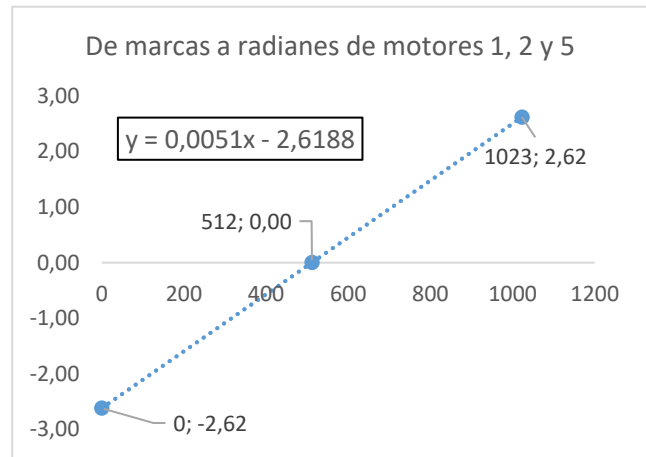


Gráfica 2 De radianes a marcas de motores 1, 2 y 5

Del mismo modo, cuando trabajemos por guiado manual, será muy útil pasar la lectura de la posición articular expresada en marcas a radianes para luego obtener la matriz homogénea correspondiente a esa configuración y poder aplicarle traslación o rotación. De nuevo, se considera funciones de tipo lineal, para los motores 3 y 4 (ver Gráfica 4), mientras que para los motores 1,2 y 5 (ver Gráfica 5).



Gráfica 5 De marcas a radianes de motores 3 y 4



Gráfica 4 De marcas a radianes de motores 1, 2 y 5

Llegados a este punto, podríamos decir que ya tenemos las funciones necesarias. Pero aún nos queda un último apartado. Y es que resulta, que las soluciones en radianes obtenidas por la cinemática inversa no tienen por qué estar dentro del rango angular que tenemos en los motores. Recordemos que las soluciones obtenidas no tienen en cuenta las limitaciones articulares.

Para ejemplificar esto, imaginemos que para el motor 1 obtenemos una solución de $\frac{23}{12}\pi$ (345°), la cual numéricamente no está dentro del rango articular (ver Figura 19). Pero, si consideramos el ángulo $\frac{-\pi}{12}$ (-15°), que es equivalente, sí que entra en el rango articular.

Este inconveniente lo tenemos para los ángulos que tienen offset y para aquellos que no tienen únicamente un atan2 como solución, por ejemplo, el ángulo θ_2 :

$$\theta_2 = \text{atan2}(V_{124}, V_{114}) \pm \arccos\left(\frac{\gamma}{+\sqrt{V_{114}^2 + V_{124}^2}}\right)$$

Para solucionar el problema mencionado deberemos expresar las soluciones en el rango $[-\pi, \pi]$.

Para ello se implementó⁵ lo siguiente:

- Hallar el $\sin(\alpha)$ y el $\cos(\alpha)$.
- Usamos la función atan2 , que devuelve el ángulo, β , en rango $[-\pi, \pi]$ y lo ubica en el cuadrante adecuado porque tiene en cuenta el signo de sus argumentos, los cuales serán el $\sin(\alpha)$ y el $\cos(\alpha)$. Para su mayor comprensión se adjunta la figura 21.

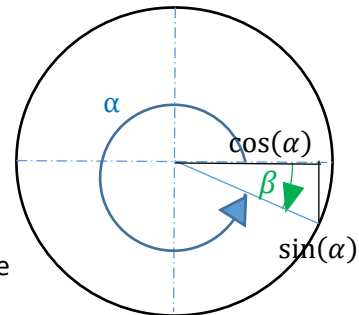


Figura 21 $\beta \in [-\pi, \pi]$

La solución anterior no es suficiente para los motores 3 y 4. Si imaginamos que la solución es de $\frac{3}{4}\pi$ (135°), entonces numéricamente no entra dentro del rango de soluciones (ver Figura 20) de estos motores. Pero si lo expresamos como $\frac{-5}{4}\pi$ (-225°), entonces sí que es una solución válida.

Para solucionar esto se implementó adicionalmente⁶ a lo anterior, para los motores 3 y 4:

- Si el ángulo obtenido, θ , es mayor que $\frac{\pi}{3}$ (60°), entonces recalculamos el ángulo θ de la siguiente manera:

$$\theta = -\pi - (\pi - \theta).$$

De esta forma conseguimos que las soluciones de los motores 3 y 4 sean expresadas en el rango $[-\frac{5}{3}\pi, \frac{\pi}{3}]$.

Hemos definido la cinemática inversa, que nos permitirá planificar trayectorias rectilíneas. Y se ha mostrado el tipo de discretización que usan los motores, y su relación con los ángulos expresados en radianes. Ahora podremos pasar a la programación del robot.

⁵ Se encuentra al final del **ikinePhantomX.m** (ver Anexo 3)

⁶ Se encuentra al final del **ikinePhantomX.m** (ver Anexo 3)

4. Programación del robot

Tras haber abordado la parte cinemática del robot, podríamos continuar con la parte dinámica; sin embargo, no lo haremos y explicaremos sus motivos a lo largo de este capítulo.

Para evitar extendernos en los aspectos técnicos de los motores se comentará aquellos que se han considerado importantes.

Apoyándonos en las librerías de Corke y la cinemática que hemos desarrollado en el capítulo anterior, realizaremos un primer tipo de experimentos donde se estudiará la evolución de la posición y velocidad. Durante este proceso, nos encontraremos con diferentes errores en las librerías de Corke que iremos solucionando hasta conseguir ejecutar con éxito este primer tipo de experimentos y conseguir sus curvas de posición y velocidad. Además, se desarrollará nuevas funciones sobre dichas librerías, con el fin de poder leer y escribir sobre más registros de control. Y para culminar, realizaremos un segundo tipo de experimentos en el que analizaremos cómo influye variar la evolución del par en la etapa transitoria del accionamiento.

4.1 Especificaciones de los motores, aspectos importantes

Las especificaciones de los motores se pueden consultar en profundidad en el manual que se encuentra en la página web de ROBOTIS e-manual [21] . No obstante, a continuación, se comentará los aspectos técnicos y de programación que se han considerado importantes y que pretenden dar una visión general antes de leer el manual y comenzar a programar. Se comentará la relación de reducción que llevan los motores, los sensores que lleva incorporados y el tipo de discretización que se usa para la posición y velocidad, el tipo de comunicación, los registros que maneja y si son de tipo volátil o no y el tipo de paquete de datos que se usa.

Los Dynamixel AX12-A incorporan un motor DC, un reductor [254:1] y un circuito de control que se comunica con la placa ArbotiX-M. A pesar de ser un pequeño motor ofrece un fuerte par de acción. La tabla con las especificaciones técnicas [6] se puede encontrar en el Anexo 4.

La posición angular está discretizada en 1024 marcas o pasos. Para identificar la marca en la que nos encontramos, el motor dispone de un sensor de posición, tal como se puede apreciar en el

circuito [22] que se adjunta en el Anexo 5. Aunque el circuito mostrado corresponde a una versión anterior al de nuestro motor, el esquema básico es el mismo.

A partir del circuito (Anexo 5), observamos que no hay un sensor para la lectura de la velocidad. Por este motivo las velocidades de los motores que devuelve la placa son solamente estimaciones. De hecho, el estudio realizado por Ethan Tira-Thompson [23], muestra que la lectura de posición es muy buena, teniendo un error de aproximadamente 0.5° , a una alta velocidad de comunicación. En cambio, la lectura de la de velocidad no sólo es imprecisa, error por encima del 10%, sino que además se actualizan cada 130 ms, lo que implica que en ocasiones tenga zonas muertas, aspecto que comprobaremos en el apartado 4.4.

La velocidad, al igual que la posición, se expresa en marcas. Su rango y valor depende del modo de funcionamiento en el que se esté trabajando. En nuestro caso, el modo sobre el que trabajaremos es el modo *joint*. El rango es 0~1023 en valor absoluto⁷, y cada marca representa 0.111 rpm, a excepción del 0, que implica que el motor se va a mover a la máxima velocidad permisible en cada momento, por lo que no existe un control de velocidad. Por tanto, la mínima velocidad será 0.111 rpm que corresponde a la marca 1 y la máxima de ~114 rpm bajo condiciones de carga nula.

Conocer el tipo de discretización que se maneja nos permitirá interpretar mejor los resultados que se observen al graficar la posición y la velocidad.

En cuanto a la comunicación es de tipo serie asíncrona con 8 bits. La velocidad de transmisión de datos entre la placa y los motores puede llegar hasta 1 Mbps y es la que usaremos. Por ejemplo, en el apartado 4.3 nosotros cargaremos el programa **pypose.ino** en la placa, en él la velocidad viene configurada a 1 Mbps.

En cambio, la velocidad que configuraremos entre el ordenador y la placa será de 38400 bps. Este parámetro lo configuraremos desde el entorno del MatLab, a la hora de realizar la comunicación con la placa, tal como podremos ver en **inicialización_comunicacion.m** (ver Anexo 3). Se ha querido aclarar estos dos aspectos, porque en la bibliografía la información puede resultar confusa en cuanto a este tema, ya que el manual de los motores está separado del de la placa.

⁷ El registro de la velocidad es **Present Speed** y el sentido de giro lo marca el bit 10 del registro **Present Load** [40].

La interacción con los motores lo realizaremos mediante la lectura y escritura sobre una tabla de control (“Control table” [24]), que contiene varios registros. La tabla contiene 2 partes bien diferenciadas, EEPROM y RAM. La primera mantiene los datos guardados en los registros tras una desconexión a la red eléctrica, mientras que la segunda no. Este aspecto es importante; por ejemplo, la velocidad se encuentra en la RAM, y su valor inicial es 0, pudiendo llegar a ser peligroso el movimiento debida a la alta velocidad. Por ello, tras volver a conectar la placa a una fuente de alimentación debemos configurar antes la velocidad que la consigna de posición.

Los paquetes de datos que usaremos para leer y escribir sobre los registros se codifican en código binario, que representaremos a través de números hexadecimales. Hay dos tipos de paquetes de datos, el de instrucciones y el de estado. El paquete de instrucciones sigue la siguiente estructura:

0XFF	0XFF	ID	LENGTH	INSTRUCTION	PARAM 1	...	PARAM N	CHECK SUM
------	------	----	--------	-------------	---------	-----	---------	-----------

Al estar 5 motores conectados a un mismo bus, no debe haber dos motores con el mismo ID, de lo contrario la instrucción no se ejecutará correctamente. El resto de los términos (length...), se deben configurar tal como se muestra en el manual de comunicación [25] o si se prefiere el manual para poder imprimirlo [26] [27].

Cualquier acción que realicemos, el ordenador recibirá un paquete de estado que seguirá la siguiente estructura:

0XFF	0XFF	ID	LENGTH	ERROR	PARAM 1	...	PARAM N	CHECK SUM
------	------	----	--------	-------	---------	-----	---------	-----------

Quizás mencionar la estructura de los paquetes de datos sea muy específico, pero nos sirven de introducción para el apartado 4.5.2 donde volveremos a retomarlos para solucionar errores en las librerías.

4.2 Librerías

En el apartado anterior comentamos brevemente acerca del paquete de instrucciones. Se puso a disposición del lector las referencias a las que puede acudir para profundizar en el tema, donde

además puede encontrar ejemplos de programas realizados a bajo nivel. Pero trabajar de este modo, puede resultar muy laborioso. Es por ello, que el fabricante proporciona librerías [3] .

Con ellas, la programación se realizará a alto nivel. Cualquier modificación o ampliación necesaria de las librerías, las podemos realizar si conocemos el código a bajo nivel.

Las librerías proporcionadas por el fabricante nos permiten preparar nuestros primeros programas sobre la plataforma Arduino. Sin embargo, estas librerías no disponen de funciones que permitan graficar la evolución de la posición o velocidad de cada uno de nuestros motores o simular el movimiento del robot. Por ello, nuestra programación se realizará sobre MatLab [4], además esto nos facilitará en un futuro incorporar cámaras como la LeapMotion, que permiten un control gestual en tiempo real mediante el movimiento de nuestras manos. De hecho, en la Universidad Nacional de Colombia se ha desarrollado el control del PhantomX Pincher a través de la LeapMotion [28].

Como comentamos en el párrafo anterior, el fabricante proporciona librerías para Arduino. Sin embargo, para MatLab no ha desarrollado nada. Por este motivo, nos apoyaremos en el trabajo realizado por el profesor Peter Corke, quien en su documentación [4], describe que el programa **pypose.ino**, proporcionado por el fabricante, permite interactuar con la placa a través del lenguaje de MatLab. Además, en la última versión de la toolbox desarrollada por él, RBT 10.3.1 (agosto 2018), se encuentra la librería, **Arbotix.m**, que establece la comunicación entre el ordenador y la placa, así como una serie de funciones básicas; por ejemplo, introducir la consigna de cualquiera de los motores, así como su velocidad sin tener que preocuparnos de escribir el paquete de instrucciones. Otra función interesante, es activar o desactivar el par resultante, que junto a la capacidad de leer la posición actual de los motores nos permitirá realizar trayectorias por guiado manual.

4.3 Conexión

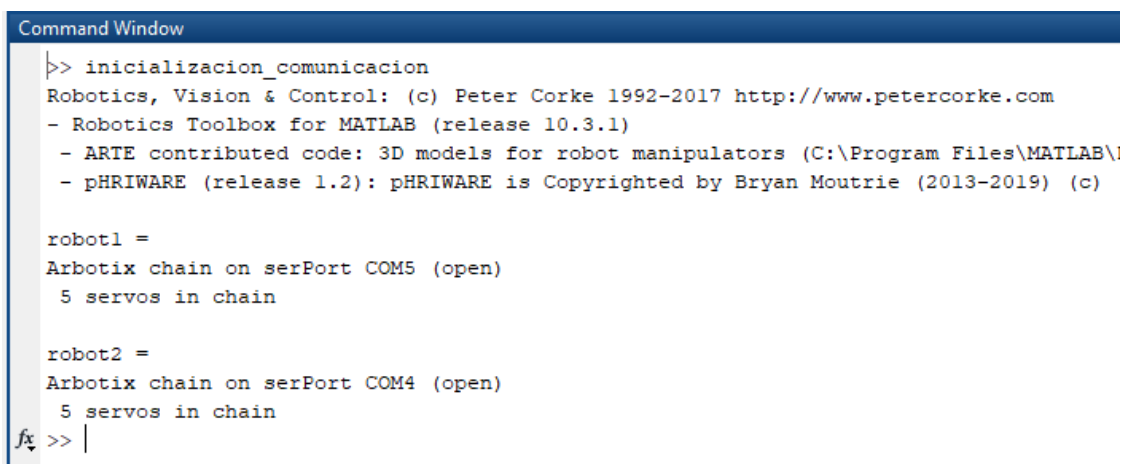
Ya hemos definido el entorno y las librerías sobre las que trabajaremos. A continuación, estableceremos la conexión entre nuestro robot y la placa junto a los motores. Para ello, seguiremos los pasos que establece Corke [4]. Primero cargamos el programa **pypose.ino** en las placas **ArbotiX-M**. Este paso se realiza una única vez, siempre y cuando no instalemos otro programa desde Arduino.

A continuación, desde MatLab establecemos la comunicación, a través de **inicializacion_comunicacion.m** (ver Anexo 3), en cuyo código se adjunta explicaciones de lo que está realizando.

Aprovecharemos para comentar un aspecto muy importante y que puede confundir al lector. En este proyecto, se trabajará con 2 versiones de toolboxes, ambos pertenecen a Corke. El motivo de esto es que, la versión antigua, RBT 9.8, se había usado en el curso de robótica y se estaba familiarizado con ella por lo que al principio se trabajó con ella. En el transcurso del proyecto se vio, que esta versión no incluía la librería **Arbotix.m**, la cual era necesaria para poder interactuar con los motores a través de MatLab. Por este motivo se planteó ejecutar todos los programas que ya habíamos hecho con la nueva versión de toolbox, RBT 10.3.1, pero resultó que había incompatibilidades de directorios y nuevas formas de llamar a las funciones. Por ello, para evitar rehacer el trabajo avanzado, se decidió trabajar con ambas toolboxes.

Lo anterior implicaba alternar la instalación de la toolbox, cada vez que vayamos a usar una u otra. Como se puede ver en el Anexo 1, se propone una forma sencilla de hacerlo, a través del uso de rutas de búsqueda. Aunque en el Anexo 1, hablamos sobre la instalación de la RBT 9.8, es totalmente aplicable para la RBT 10.3.1, tal como se puede apreciar en la primera línea de código de **inicializacion_comunicacion.m**.

Si la conexión se ha realizado correctamente deberá aparecer el cuadro de mensaje que se observa en la figura 22 sobre la ventana de comandos de MatLab. A continuación, podremos usar las funciones predefinidas de la librería **Arbotix.m**.



```
Command Window
>> inicializacion_comunicacion
Robotics, Vision & Control: (c) Peter Corke 1992-2017 http://www.petercorke.com
- Robotics Toolbox for MATLAB (release 10.3.1)
- ARTE contributed code: 3D models for robot manipulators (C:\Program Files\MATLAB\
- pHRIWARE (release 1.2): pHRIWARE is Copyrighted by Bryan Moutrie (2013-2019) (c)

robot1 =
Arbotix chain on serPort COM5 (open)
  5 servos in chain

robot2 =
Arbotix chain on serPort COM4 (open)
  5 servos in chain
fx >> |
```

Figura 22 Respuesta ante una ejecución correcta de **inicializacion_comunicacion.m**

4.4 Experimentos para evaluar la evolución de la posición y velocidad

Ahora que hemos realizado la conexión, continuamos con un primer tipo de experimentos en el que vamos a ver la evolución de la posición y velocidad de cada una de las articulaciones con el objetivo de comprobar y cuantificar el error de posición. Además, se buscará comprobar los puntos muertos que había al leer la velocidad [23].



Figura 23 Configuración $q_1=[512\ 512\ 512\ 512\ 512]$ (izquierda) y configuración $q_2=[600\ 600\ 600\ 600\ 600]$ (derecha)

Este primer movimiento va a ser muy sencillo en el que moveremos uno de los robots de $q_1 = [512\ 512\ 512\ 512\ 512]$ a $q_2 = [600\ 600\ 600\ 600\ 600]$ (ver Figura 23) a una velocidad de consigna de 100 (ver Figura 24). Y no se ha modificado ningún otro registro a parte de estos.

Las líneas de comando principales son las que se pueden observar en la figura 24. El resto de código que se emplea para recoger los datos de posición y velocidad se muestra en el Anexo 3, bajo el nombre de **experimento_tipo1.m**.

```
q2 = [600 600 600 600 600];  
v_1 = 100*[1 1 1 1 1];  
robot1.setpos(q2,v_1);
```

Figura 24 Código para alcanzar la posición q_2 a una velocidad v_1

Como se puede observar en las líneas de comando, no se utiliza ninguna instrucción de par. Y es que a pesar de las buenas prestaciones que ofrecen estos motores, no nos permiten realizar un control sobre el par motor, sino que se realiza internamente.

Y aunque existen parámetros relacionados con el par que se pueden configurar como el *compliance margin* [29], *compliance slope* [30] y *punch* [31] sólo sirven para conseguir un movimiento más fino si disminuimos la pendiente de actuación del par, tal como se puede observar en la figura 25. Además, siempre tendremos que asumir un error de posición [29] que se configura con el *compliance margin* (ver Figura 25). Este registro es volátil y siempre se reinicia a 1, que es equivalente a $\pm 0.29^\circ$. Con la obtención de la evolución de las gráficas de posición podremos observar si el error observado experimentalmente se corresponde con el valor configurado en el registro de *compliance margin*.

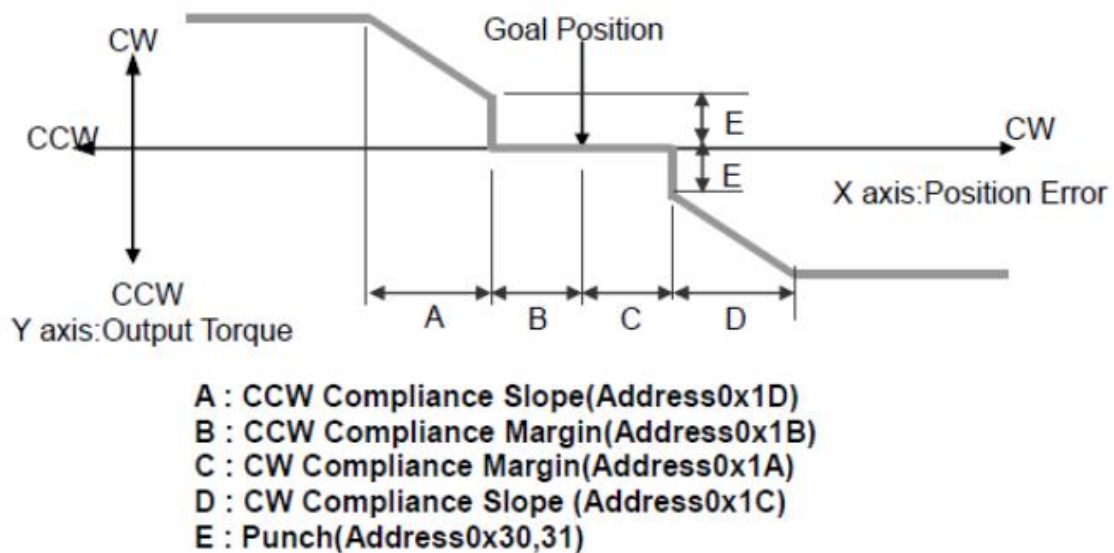
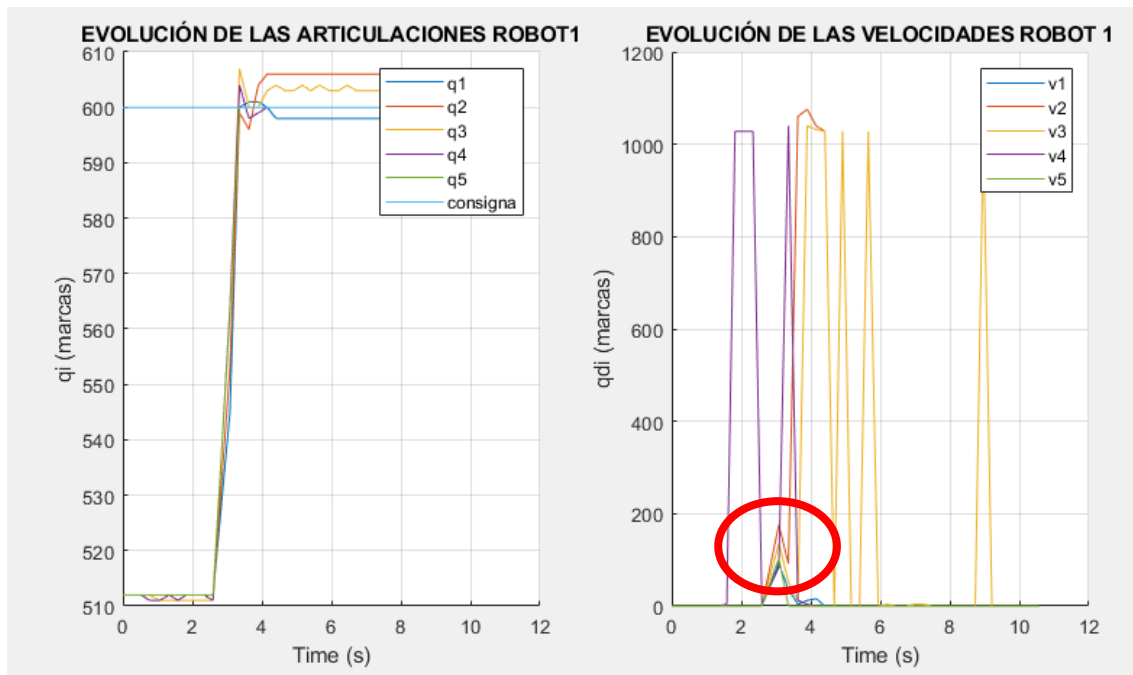


Figura 25 Parámetros de control dinámico en un Dynamixel AX-12 [30]

En la gráfica 6 se muestra el resultado de la evolución de la posición y de la velocidad de un movimiento en el que se ha configurado una velocidad de 100 marcas (≈ 11.1 rpm) y no se ha modificado ningún parámetro relacionado con el par (ver Figura 25). Este primer tipo de experimento se realizó sin carga un número repetido de veces, y los resultados siempre eran parecidos en cuanto a posición, pero en velocidad diferían los picos. En algunos casos, salían más y en otros menos y su posición a lo largo del tiempo era diferente.



Gráfica 6 Evolución de posición y velocidad en un primer tipo de experimento sobre un único robot.

En cuanto a la velocidad, no se observan zonas muertas, pero sí se observa que hay picos de velocidad sobre todo en torno al movimiento con velocidad cuasi-constante (ver círculo rojo), donde todos los motores tienden a tener una velocidad de 100 marcas. Esto quizás se deba a un movimiento de vaivén, tal como le ocurre a la articulación 3 al llegar a la posición final, y que se traduce en picos de velocidad.

Para evitar los vaivenes, podríamos disminuir la pendiente de desaceleración de par, disminuyendo con ello la desaceleración angular. Pero, la librería de **Arbotix.m** no dispone de funciones que nos permitan leer y escribir los parámetros de *compliance margin* [29], *compliance slope* [30] y *punch* [31]. Por ello, en el apartado 4.5 desarrollaremos nuevas funciones de lectura y escritura sobre estos y otros registros más.

Por otro lado, se observa un error de posición en todos los motores incluso sin carga. El error que se observa para algunas articulaciones es mayor que 1 marca (que equivale a $\pm 0.29^\circ$), cuyos valores no coinciden con los valores configurados de fábrica sobre el registro de *compliance margin*. De hecho, investigadores de la International Federation of Automatic Control, IFAC, realizaron un informe [32] en el que comentan esto y desarrollan una solución de control dinámico para eliminar el error de posición, pero para un solo motor donde obtienen soluciones muy buenas incluso con carga. Y dejan como tarea aplicarlo a los 5 motores, cuyo inconveniente, es el alto acoplamiento. No se realizará esto porque implicaría una mayor extensión del proyecto.

Para detectar el final de movimiento, se comparará la posición actual con la consigna. Cuando la diferencia en todas las articulaciones sea menor a un error admisible que fijaremos nosotros, entonces se considerará que el movimiento ha finalizado. Tras varias ejecuciones de este primer movimiento se ha observado que el error siempre es inferior a 10 marcas. Por este motivo, definiremos que el error admisible será de 10 marcas, que es equivalente a $\pm 2.90^\circ$, para futuros movimientos.

4.5 Modificación de la librería Arbotix.m

En el apartado anterior, analizamos las curvas de posición y velocidad de un movimiento experimental. Pero antes de obtener esas curvas, tuvimos varias dificultades al ejecutar dicho movimiento, por lo que a continuación, nombraremos cada uno de los problemas y sus correspondiente soluciones que hemos desarrollado. También, añadiremos más funciones de lectura y escritura sobre otros registros.

4.5.1 Corrección y modificación de la función `arb.setpos(id,pos,speed)`

`Arb.setpos(id,pos,speed)` es una de las funciones que está implementada en la librería **Arbotix.m**. Esta función nos permite alcanzar la posición deseada para cada uno de los motores a la velocidad que nosotros queramos.

Si recordamos, la velocidad está en la RAM, y su valor es volátil; es decir, se reinicia a cero (que equivale a la velocidad más alta posible) al desconectar la placa. Por lo que, cuando configurábamos una consigna de posición y velocidad tras conectar la placa a su fuente de alimentación, el robot se movía a una velocidad muy alta. Sin embargo, si volvíamos a ejecutar otra orden de movimiento a la misma velocidad configurada anteriormente, entonces el robot lo hacía correctamente.

Por lo tanto, revisamos el código de **Arbotix.m**, donde se observó que la estructura del código no era el correcto. El problema se debía a que se escribía primero la posición en el registro de control, y luego la velocidad, tal como se puede observar en la figura 26.

```
% single joint mode
id = varargin{1}; pos = varargin{2};

arb.writedata2(id, Arbotix.ADDR_GOAL, Arbotix.a2e( pos ));

if nargin == 4
    speed = varargin{3};
    arb.writedata2(id, Arbotix.ADDR_SPEED, speed);
end
```

Figura 26 Parte del código `arb.setpos(varargin)`, versión original de Peter Corke

Para solucionarlo, invertimos el orden (ver Figura 27). Primero configuramos la consigna de velocidad a la que se van a mover cada una de las articulaciones. De ese modo el registro que contiene la consigna de velocidad, **moving speed** [33], queda grabado con el nuevo valor que hemos puesto. A continuación, se escribe la consigna de posición sobre el registro **goal position** [34]. Una vez que se ha escrito la consigna de posición el robot comienza a moverse e intentará alcanzar la velocidad de consigna (ver Gráfica 6, círculo rojo).

```
% single joint mode
id = varargin{1}; pos = varargin{2};

%Write first speed
if nargin == 4
    speed = varargin{3};
    arb.writedata2(id, Arbotix.ADDR_SPEED, speed);
end

% Now write pose
arb.writedata2(id, Arbotix.ADDR_GOAL, pos);
```

Figura 27 Parte del código `arb.setpos(varargin)`, versión corregida

Como comentamos al principio de este subapartado, la función `setpos()` requiere que le introduzcamos como parámetro la consigna de posición. En la versión original de la librería, la posición debía ser expresada en radianes. El cambio de radianes a marcas de motor, lo realizaba internamente con la función `a2e` (Ver Figura 26). Pero las ecuaciones de esa transformación están basadas sobre otra posición diferente de reposo a la nuestra, concretamente su posición de reposo es [512 512 512 512 512] mientras que la nuestra es [512 512 814 814 512], lo que hace que las ecuaciones sean diferentes para los motores 3 y 4; tal como se vio en el apartado 3.3.

Por ello, nosotros al escribir la función `setpos()` lo haremos con ángulos expresados en marcas, de manera que las transformaciones lo haremos externamente. Del mismo modo, para su lectura usaremos marcas.

4.5.2 Modificación de la función `receive(arb)`

Para obtener las gráficas de posición, necesitamos leer reiteradamente la posición de cada uno de los motores, tal como se plantea en el código `experimento_tipo1.m` (ver Anexo 3). Sin embargo, en la ventana de comandos siempre aparecía la advertencia que podemos ver en la figura 28, y no conseguíamos obtener la evolución de la posición y velocidad.

Los dos primeros bytes indican el comienzo del paquete de datos. El ID corresponde al identificador del motor. El byte *length* es igual al número de parámetros más 2 unidades, que corresponden al byte *error* y *check sum*. Teniendo esto en cuenta, si analizamos los bytes que se reciben según la figura 30; el tercer byte vale 2 y corresponde a *length*; por lo cual, el número de parámetros se queda en cero, que obviamente no tiene sentido; ya que los parámetros contienen el valor numérico de la posición. Evidentemente esto significa que no se ha realizado la lectura correctamente, ya que tras el tercer byte sólo aparece el byte de *error* y *check sum*; y se repite el mismo problema con los siguientes motores hasta que se llena el buffer, salta el tiempo de período y no se habrá decodificado ninguna posición ya que no se dispone de valor para decodificar.

Para solucionar esto, fuimos a la función *receive* que se encuentra dentro de la librería **Arbotix.m**. Esta función, se encarga de decodificar el paquete de datos que se recibe en código binario. Se trata básicamente de una máquina de estados en la que va avanzando conforme va leyendo byte a byte, tal como podemos ver en la figura 31.

En la versión original, pasa del estado 3 al siguiente estado independientemente del valor de *count*, que es el número de parámetros. Si recordamos el problema estaba relacionado

```

switch state
  case 0 % expecting first 0xff
    if c == 255
      state = 1;
    end
  case 1 % expecting second 0xff
    if c == 255
      state = 2;
    end
  case 2 % expecting id
    s.id = c;
    state = 3;
  case 3 % expecting length
    len = c;
    count = len-2;
    params = [];
    state = 4;
end

switch state
  case 0 % expecting first 0xff
    if c == 255
      state = 1;
    end
  case 1 % expecting second 0xff
    if c == 255
      state = 2;
    end
  case 2 % expecting id
    s.id = c;
    state = 3;
  case 3 % expecting length
    len = c;
    count = len-2;
    %It's necessary this
    %conditional statement(from
    %Josué David Poma 20/07/2019)
    if count ~= 0
      params = [];
      state = 4;
    else
      state = 0;
    end
end

```

Figura 31 Máquina de estados para decodificar el paquete de estado, versión original (izquierda) y versión modificada (derecha)

justamente con esto, que el número de parámetros era cero, y el decodificador continuaba leyendo nuevos paquetes.

Para solucionar esto, añadimos la sentencia condicional que se puede ver en la versión modificada (Figura 31, derecha), donde ordenamos que, si *count* es nulo, entonces volvemos al estado inicial. De esta manera logramos solucionar el problema planteado anteriormente.

4.5.3 Adición de más funciones de lectura y escritura.

En el apartado 4.4 mostramos una gráfica de velocidad sin entrar en detalles sobre cómo se llevan a cabo dichas lecturas. La librería original traía la función de leer **moving speed** que, si recordamos (ver apartado 4.5.1), es el registro en el que se guarda la consigna de velocidad. Sin embargo, la librería no traía función para leer **present speed** [35], cuyo registro contiene la velocidad actual mediante estimación, porque debemos recordar que los motores no tienen sensor de velocidad. Por este motivo, desarrollamos la función de lectura de velocidad `arb.getcurspeed(id)`, y que implementamos en la librería **Arbotix.m** (ver Anexo 3).

Del mismo modo, en el apartado 4.4 se habló de que existe un regulador de la curva de par aplicada principalmente a través del *compliance slope* [30]. Así también, se nombró otros parámetros como el *compliance margin* [29] y *punch* [31]. Con la librería original no podíamos leer ni escribir sobre dichos registros; así que desarrollamos nuevas funciones de lectura y escritura, que se implementaron en la librería (**Arbotix.m**, ver Anexo 3):

Lectura	Escritura
<code>arb.getcwcompmar(id)</code>	<code>arb.setcwmargin</code>
<code>arb.getcwwcompmar(id)</code>	<code>arb.setccwmargin</code>
<code>arb.getcwcomslope(id)</code>	<code>arb.setcwslope</code>
<code>arb.getcwwcomslope(id)</code>	<code>arb.setcwwslope</code>
<code>arb.getpunch(id)</code>	<code>arb.setpunch</code>
<code>arb.getcurspeed(id)</code>	

4.6 Experimentos de ajuste del perfil de movimiento

Ahora que hemos implementado nuevas funciones sobre la librería **Arbotix.m** (ver Anexo 3) y corregido algunas funciones, vamos a realizar un segundo tipo de experimentos, en el que veremos cómo afecta al movimiento la modificación del *compliance slope* [30]. Pero antes, realizaremos un programa **obtención_datos.m** (ver Anexo 3), que nos muestra el valor actual

tanto de los registros que ya tenían su función de lectura en la librería original como de los registros a los que vamos a leer por primera vez tras haber creado sus funciones de lectura correspondiente. Se aprovechó también la lectura de otros registros como la temperatura. La respuesta a ese código lo podemos ver en la figura 33.

El valor del *compliance slope* es de 32 (ver Figura 33) tanto en sentido de las agujas del reloj (clockwise, cw) como en el sentido antihorario (counter clockwise, ccw) para ambos robots. Este valor se encuentra en la RAM por lo cual es volátil y se reinicia al valor de fábrica que es 32. Aunque puede tomar valores desde 0 a 255; en realidad, sólo hay 7 pasos o configuraciones para el *compliance slope* tal como se puede observar en la figura 32.

Step	Data Value	Data Representative Value
1	0(0x00) ~ 3(0x03)	2(0x02)
2	4(0x04) ~ 7(0x07)	4(0x04)
3	8(0x08)~15(0x0F)	8(0x08)
4	16(0x10)~31(0x1F)	16(0x10)
5	32(0x20)~63(0x3F)	32(0x20)
6	64(0x40)~127(0x7F)	64(0x40)
7	128(0x80)~254(0xFE)	128(0x80)

Figura 32 Modos de configuración del compliance slope [30]

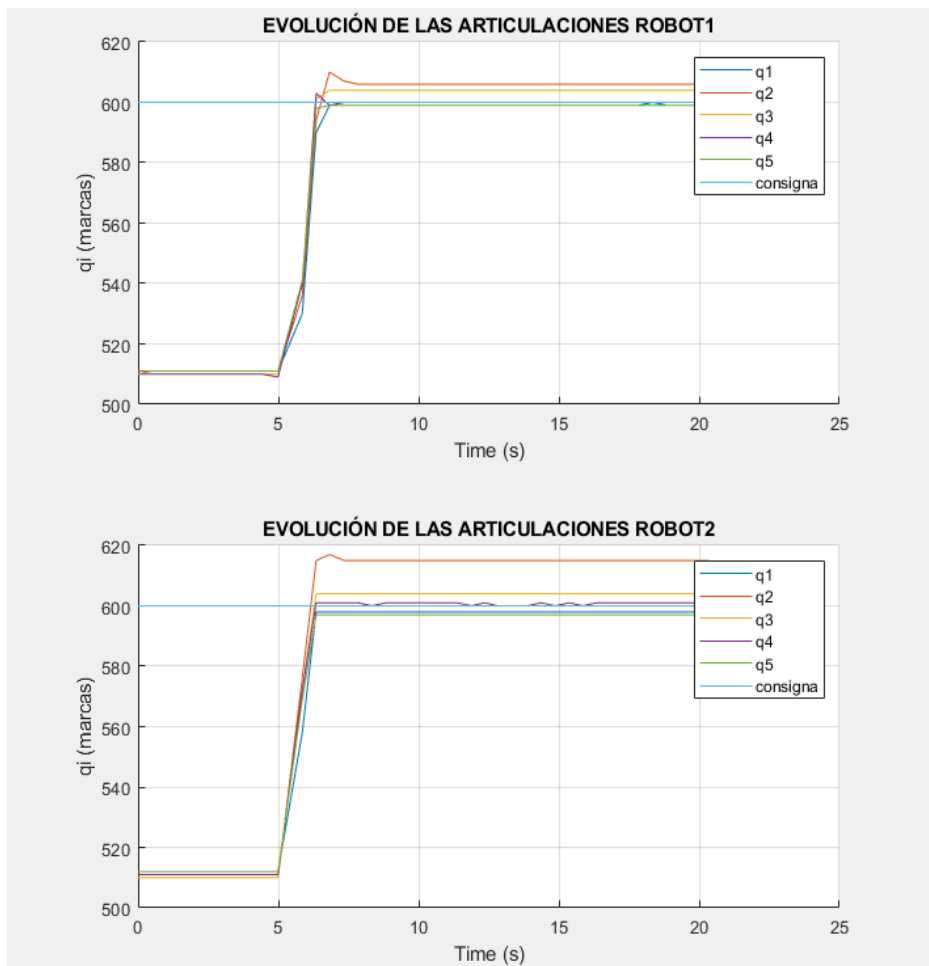
```

>> obtencion_datos
DATOS:
R1_CW COMPLIANCE MARGIN :      1      1      1      1      1
R2_CW COMPLIANCE MARGIN :      1      1      1      1      1
-----
R1_CCW COMPLIANCE MARGIN :      1      1      1      1      1
R2_CCW COMPLIANCE MARGIN :      1      1      1      1      1
-----
R1_CW COMPLIANCE SLOPE :      32      32      32      32      32
R2_CW COMPLIANCE SLOPE :      32      32      32      32      32
-----
R1_CCW COMPLIANCE SLOPE :      32      32      32      32      32
R2_CCW COMPLIANCE SLOPE :      32      32      32      32      32
-----
R1_SETTING SPEED :            0      0      0      0      0
R2_SETTING SPEED :            0      0      0      0      0
-----
R1_PUNCH :                    32      32      32      32      32
R2_PUNCH :                    32      32      32      32      32
-----
R1_TEMPERATURE :              38      54      53      37      40
R2_TEMPERATURE :              37      54      53      38      40
-----
R1_CURRENT POSE (encoder) :    512      515      816      812      512
R2_CURRENT POSE (encoder) :    510      513      817      814      510
-----
R1_CURRENT POSE (degree) :     0.15      1.03      -0.7      -1.88      0.15
R2_CURRENT POSE (degree) :    -0.44      0.44      -0.41      -1.29      -0.44
-----
R1_CURRENT POSE (radian) :      0      0.02      -0.01      -0.03      0
R2_CURRENT POSE (radian) :    -0.01      0.01      -0.01      -0.02      -0.01
-----
R1_CURRENT SPEED :            0      0      0      0      0
R2_CURRENT SPEED :            0      0      0      0      0

```

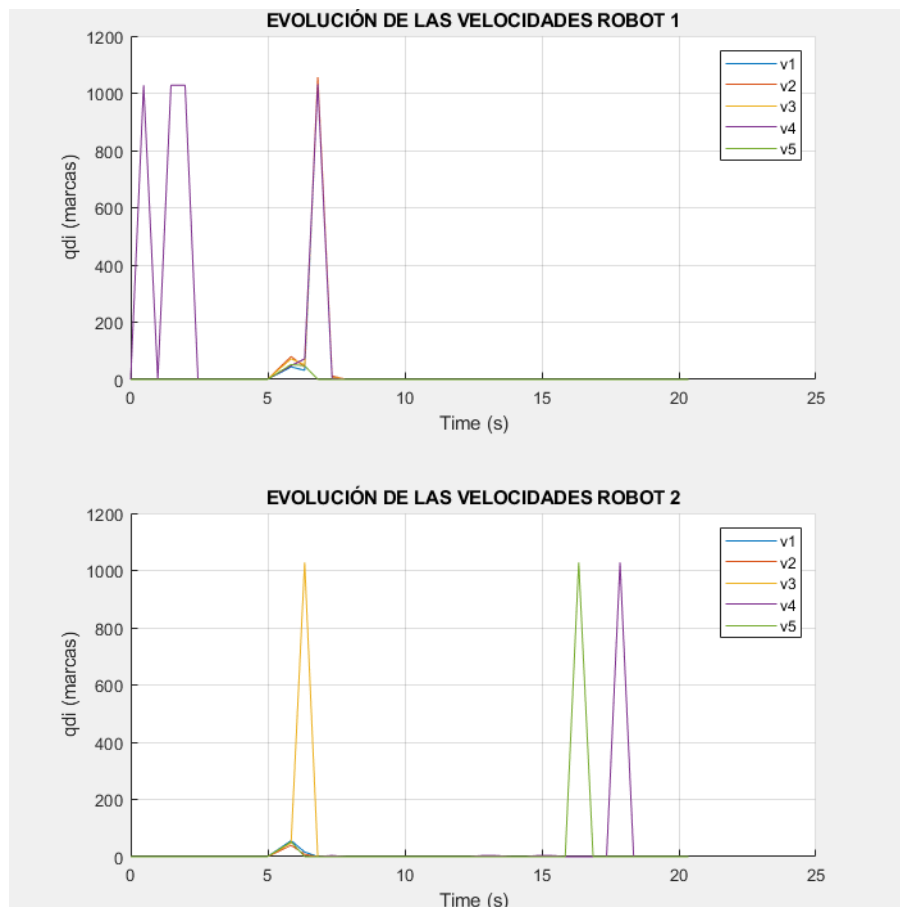
Figura 33 Valor de varios registros de ambos robots.

El segundo tipo de experimentos va a ser similar al primero (ver apartado 4.4), el robot partirá de $q_1 = [512 \ 512 \ 512 \ 512 \ 512]$ a $q_2 = [600 \ 600 \ 600 \ 600 \ 600]$ con una velocidad de consigna de 100. Pero en esta ocasión lo haremos sobre los dos robots y configuraremos un *compliance slope* diferente para ambos robots. Teniendo en cuenta lo del párrafo anterior, el robot1 tendrá un *compliance slope* de 32 (valor de fábrica), mientras que el robot2 lo configuraremos con un valor de 64 (menor pendiente); esto implica que la pendiente se reduce a la mitad, pues el punch lo mantenemos de fábrica. Lo anterior se implementó en **experimento_tipo2.m** (ver Anexo 3). La ejecución de dicho programa se realizó varias veces sobre ambos robots sin carga y el resultado de dicho movimiento en cuanto a posición y velocidad eran parecidos. Las gráficas de posición y velocidad de ambos robots se pueden ver en las gráficas 7 y 8, respectivamente.



Gráfica 7 Evolución de la posición de ambos robots moviéndose con diferente *compliance_slope*.

Tal como esperábamos, se observó experimentalmente que el movimiento es más suave y fino sobre el robot2 ya que la pendiente de actuación del par era la mitad. Pero, esto causó que el error de posición sea mayor en comparación al que se tiene en el robot1. Numéricamente dicho



Gráfica 8 Evolución de la velocidad de ambos robots moviéndose con diferente *compliance_slope*.

error está próximo a las 20 marcas, lo que equivale a un error de $\pm 5,8^\circ$; que para el tipo de aplicaciones de sujetar y desplazar objetos es suficiente. Así que, a partir de ahora usaremos en las aplicaciones un error admisible de 20 marcas, en vez de 10 (que mencionamos en el apartado 4.4).

Tras estos dos tipos de experimentos hemos solucionado los problemas que presentaba la librería original y se ha conseguido un movimiento más suave sin oscilaciones.

Así también, hemos definido un error admisible de ± 20 marcas para detectar la finalización de movimiento. Su valor vino determinado al reducir la pendiente de aplicación del par, ya que pasó un error admisible de ± 10 marcas a ± 20 marcas, pero a cambio se conseguía movimientos más suaves. El siguiente paso es desarrollar la aplicación con la suela de un calzado.

5. Aplicación

Ahora que ya hemos realizado las modificaciones y correcciones necesarias sobre la librería, y que hemos visto que podemos conseguir movimientos más suaves y que su efecto sobre el error de posición no es significativo para el tipo de aplicación que vamos a desarrollar, podemos empezar a desarrollarla.

5.1 Objetivo

Nuestro objetivo es realizar un giro de 90° a la suela de la zapatilla; es decir, pasar de la posición 1 (ver Figura 34) a la posición 2 (ver Figura 35). Realizar esta tarea con un único robot es imposible, porque carecemos del giro de rotación en el eje z de la herramienta (ver Figura 36). De hecho, si observamos la figura 36, es imposible sujetar la pieza orientada de esa manera con un robot de 4 GDL.

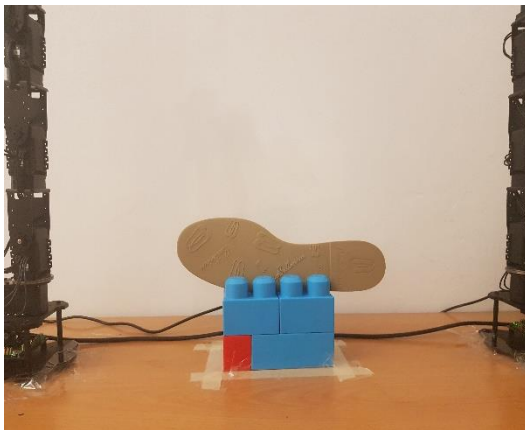


Figura 34 Posición 1

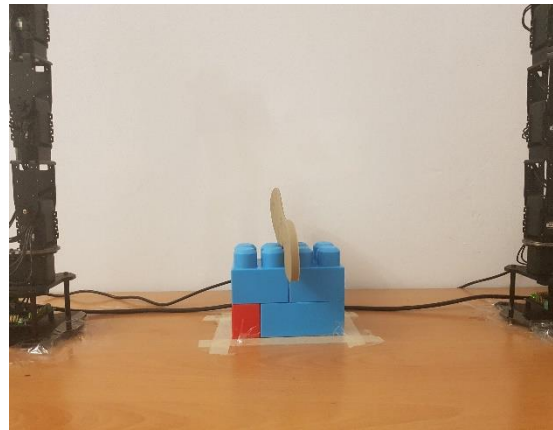


Figura 35 Posición 2

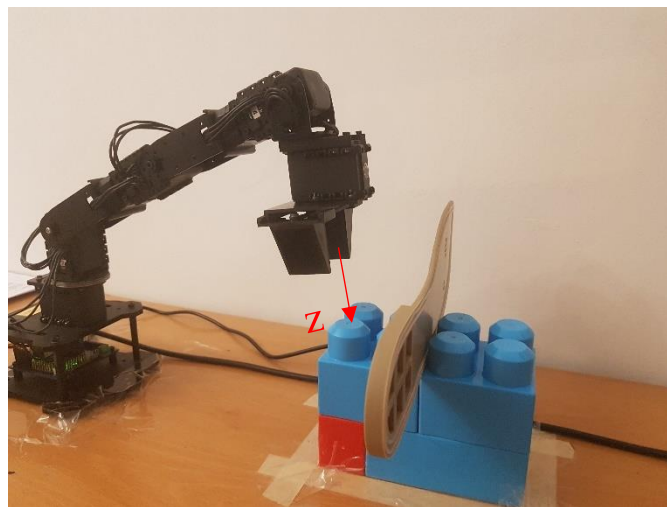


Figura 36 Incapacidad de giro en el eje z, e incapacidad de sujetar la suela.

Aprovechando que nuestro sólido es deformable y que disponemos de dos brazos robóticos, nos planteamos pasar de la posición 3 (ver Figura 37) a la posición 4 (ver Figura 38). Una vez alcanzada esa posición, nos desplazaremos hacia abajo y ayudados por las “guías” de los legos conseguiremos que la suela haya rotado.



Figura 37 Posición 3



Figura 38 Posición 4

Junto a este movimiento principal, previamente realizaremos un movimiento de recogida de la suela en su posición original, luego lo elevaremos en línea recta, después la desplazaremos de derecha a izquierda, volveremos a bajar y subir; para finalmente realizar el movimiento de rotación de la suela.

5.2 Planificación

Como hemos venido comentando a lo largo de la memoria, es necesario detectar la finalización de movimiento para poder concatenar consignas. Para dicha detección, necesitaremos leer la posición, obtener la diferencia existente en valor absoluto para cada una de las articulaciones entre el valor actual de la posición y el de la consigna, y a continuación escoger el valor máximo. Si dicho valor es menor que un error de posición admisible, entonces el movimiento ha finalizado. En caso contrario, se repetirá el proceso anterior hasta que se cumpla que el error máximo sea menor al error admisible.

Recordemos que siempre existe un error de posición, tal como se pudo observar en la gráfica 6, donde dado una misma consigna para todas las articulaciones, los motores no la alcanzaban. Por esto motivo se definió un error admisible, cuyo valor será de 20 marcas ($\pm 5.8^\circ$), y que usaremos en **aplicacion.m** (ver Anexo 3). No confundir este valor con el del *compliance margin*. Éste último, configura el error de posición que queremos tener, pero como hemos visto en el apartado 4.4, el error experimental es mayor a este valor, por ello, para detectar el final de

carrera usaremos un valor de error admisible por encima del error experimental que se observe. De hecho, cuando los robots cargan la suela, su error es mayor, teniendo que reajustar el valor de error admisible hasta 60 marcas (+/- 17°). Esto se puede ver claramente en el código **aplicacion.m** que se encuentra en el anexo 3, donde la variable que almacena el valor de error admisible se designa como “N_marcas” y se verá cómo hemos cambiado su valor en función de si estamos en un movimiento con carga o no.

Ahora que hemos explicado cómo detectamos el final de carrera, se explicará cómo se obtuvo las consignas que concatenaremos para formar una trayectoria. Cada trayectoria lo iniciaremos con un guiado manual para obtener la primera consigna. Para ello, desactivaremos el par, en caso de estar activado, mediante la función *relax(arb,varargin)*, entonces podremos mover el robot manualmente hacia la posición que nosotros deseemos, leer dicha posición y guardarla.

La mayoría de nuestras trayectorias son básicamente líneas rectas, por lo que a partir del primer punto que hemos obtenido del guiado manual, aplicaremos la cinemática de la que hemos hablado en el capítulo 3, que explicaremos paso a paso a continuación y se implementará en **planificación_recta.m** (ver Anexo 3).

De la consigna que hemos guardado nos quedamos con los cuatro primeros términos (recordemos que el último término corresponde al motor de abrir y cerrar pinza que no es un grado de libertad). Dado que está expresada en marcas lo pasamos a radianes, usando las ecuaciones del apartado 3.3, y aplicamos la cinemática directa, con la que obtenemos las matrices homogéneas correspondientes que están expresadas en metros. Por ejemplo:

$$q1 = [517 \ 499 \ 744 \ 602 \ 512] \quad \rightarrow \quad q1 = [517 \ 499 \ 744 \ 602] \quad \rightarrow$$

$$T1 = \begin{pmatrix} 0.01 & 0.03 & 1 & 0.2 \\ 0 & -1 & 0.03 & 0.01 \\ 1 & 0 & -0.01 & 0.15 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Con esa matriz, nosotros sabemos que nos encontramos a una altura de 150 mm en los ejes de la base mundo, que en nuestro caso corresponden con la base robot, B0 (ver Figura 16), porque hemos usado la matriz identidad como matriz de paso entre ambas bases. Teniendo eso en cuenta, usando una regla, medimos el desplazamiento necesario según el eje z de la base robot. En nuestro caso hemos obtenido, que debemos descender 172 mm. Si dividimos en 20 puntos, obtenemos que debemos bajar 0.0086 metros en cada paso. Por tanto:

$$T1 = \begin{pmatrix} 0.01 & 0.03 & 1 & 0.2 \\ 0 & -1 & 0.03 & 0.01 \\ 1 & 0 & -0.01 & 0.15 \\ 0 & 0 & 0 & 1 \end{pmatrix} \rightarrow T2 = \begin{pmatrix} 0.01 & 0.03 & 1 & 0.2 \\ 0 & -1 & 0.03 & 0.01 \\ 1 & 0 & -0.01 & 0.1414 \\ 0 & 0 & 0 & 1 \end{pmatrix} \rightarrow$$

$$T3 = \begin{pmatrix} 0.01 & 0.03 & 1 & 0.2 \\ 0 & -1 & 0.03 & 0.01 \\ 1 & 0 & -0.01 & 0.1328 \\ 0 & 0 & 0 & 1 \end{pmatrix} \dots T21 = \begin{pmatrix} 0.01 & 0.03 & 1 & 0.2 \\ 0 & -1 & 0.03 & 0.01 \\ 1 & 0 & -0.01 & -0.0206 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Una vez que tenemos todas las transformaciones de paso, debemos obtener los ángulos necesarios para alcanzar dichas localizaciones. Para ello, aplicamos la cinemática inversa, apartado 3.2.4, y obtenemos los ángulos en radianes (ver Figura 39). Pero lo que nos interesa es que estén expresados en marcas, usamos de nuevo las ecuaciones del apartado 3.3. Pero, además debemos añadirle la posición angular del quinto motor, que en el caso de pinza abierta es de 512. Una vez realizado todo esto, ya tenemos los puntos de paso (ver Figura 39) expresados en marcas. Debemos comprobar que todas las marcas estén dentro del rango de movimiento del robot; es decir, no tengan un valor superior a 1023 o inferior a 0.

0.0307	-0.0615	-0.3845	-1.1106	518	500	739	599	512
0.0307	-0.0948	-0.2666	-1.1952	518	493	762	582	512
0.0307	-0.1207	-0.1589	-1.2770	518	488	783	566	512
0.0307	-0.1399	-0.0595	-1.3572	518	485	802	551	512
0.0307	-0.1527	0.0328	-1.4367	518	482	820	535	512
0.0307	-0.1591	0.1189	-1.5164	518	481	837	520	512
0.0307	-0.1591	0.1993	-1.5968	518	481	853	504	512
0.0307	-0.1524	0.2743	-1.6785	518	482	867	488	512
0.0307	-0.1387	0.3441	-1.7620	518	485	881	472	512
0.0307	-0.1175	0.4086	-1.8477	518	489	893	456	512
0.0307	-0.0883	0.4677	-1.9360	518	495	905	438	512
0.0307	-0.0506	0.5211	-2.0271	518	502	915	421	512
0.0307	-0.0041	0.5686	-2.1211	518	511	924	403	512
0.0307	0.0515	0.6097	-2.2178	518	522	932	384	512
0.0307	0.1163	0.6439	-2.3167	518	535	939	365	512
0.0307	0.1898	0.6707	-2.4171	518	549	944	345	512
0.0307	0.2715	0.6897	-2.5178	518	565	948	326	512
0.0307	0.3602	0.7005	-2.6173	518	582	950	306	512
0.0307	0.4545	0.7029	-2.7140	518	601	950	287	512
0.0307	0.5526	0.6969	-2.8060	518	620	949	270	512
0.0307	0.6528	0.6825	-2.8919	518	639	946	253	512

Figura 39 Puntos de paso de una trayectoria rectilínea expresados en radianes (izquierda) y en marcas (derecha).

Tras comprobar que efectivamente, ningún valor de las marcas se encuentra fuera del rango articular de movimiento, se deberá realizar una simulación de movimiento, para comprobar que se ha conseguido una trayectoria rectilínea.

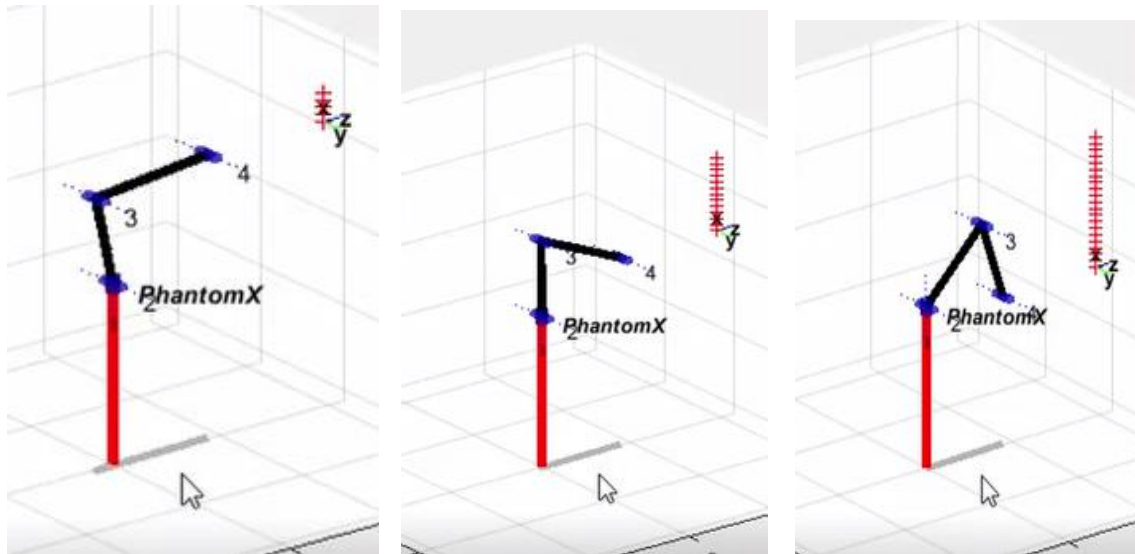


Figura 40 Simulación de trayectoria rectilínea sobre MatLab.

Tal como se puede observar en la figura 40, la trayectoria rectilínea se ha realizado correctamente. Todos los pasos que hemos mencionado están implementados en **planificación_recta.m** (ver Anexo 3). Como hemos dicho anteriormente, la mayoría de nuestras trayectorias son líneas rectas, por lo que se repetirá todo este proceso.

Los robots por defecto realizan un movimiento articular, pero los robots industriales suelen incluir funciones que les permiten realizar trayectorias rectilíneas con sólo introducirle dos localizaciones. Nuestro robot, incluye librerías para poder realizar este tipo de funciones, pero solamente para Arduino. Por ello, en MatLab recurrimos a hallar puntos de paso que estén lo suficientemente cerca, mediante cinemática inversa, y los introducimos como consignas.

Para pasar de la posición 3 a la posición 4 (ver Figura 41), el movimiento ya no es rectilíneo. Sino que, está compuesto por un movimiento de rotación en el eje z de la base del robot, B0 (ver Figura 16), y de un movimiento radial hacia el exterior manteniendo la cota z de las pinzas. Ambos movimientos se deben ejecutar a la vez porque estamos manejando un sólido que impide que desglosemos ambos movimientos. Por este motivo, el tipo de trayectoria es elíptica que podríamos planificar mediante varios puntos de paso, pero el movimiento del brazo hacia el exterior (mayor error de posición por tener más brazo en voladizo), así como las fuerzas que ejerce el sólido deformable sobre el robot, hace que planificar una trayectoria sin tener en



Figura 41 Movimiento desde la posición 3 (izquierda) a la posición 4 (derecha)

cuenta estos aspectos tenga mayor error de posición, por eso, se elige trazar la trayectoria mediante guiado manual.

5.3 Ejecución

Una vez halladas todas las consignas, las implementamos en **aplicación.m** (ver Anexo 3). Como comentamos en el apartado 4.6, el movimiento lo realizaremos con un *compliance slope* igual a 64, y manteniendo el *punch* de fábrica; lo que equivale a disminuir la pendiente a la mitad de la que viene por defecto (ver Figura 25), que conceptualmente es tener una menor des/aceleración angular y, por tanto, un movimiento más suave.

A continuación, se muestra una serie de fotos que muestran el movimiento en orden cronológico y que describiremos al final:

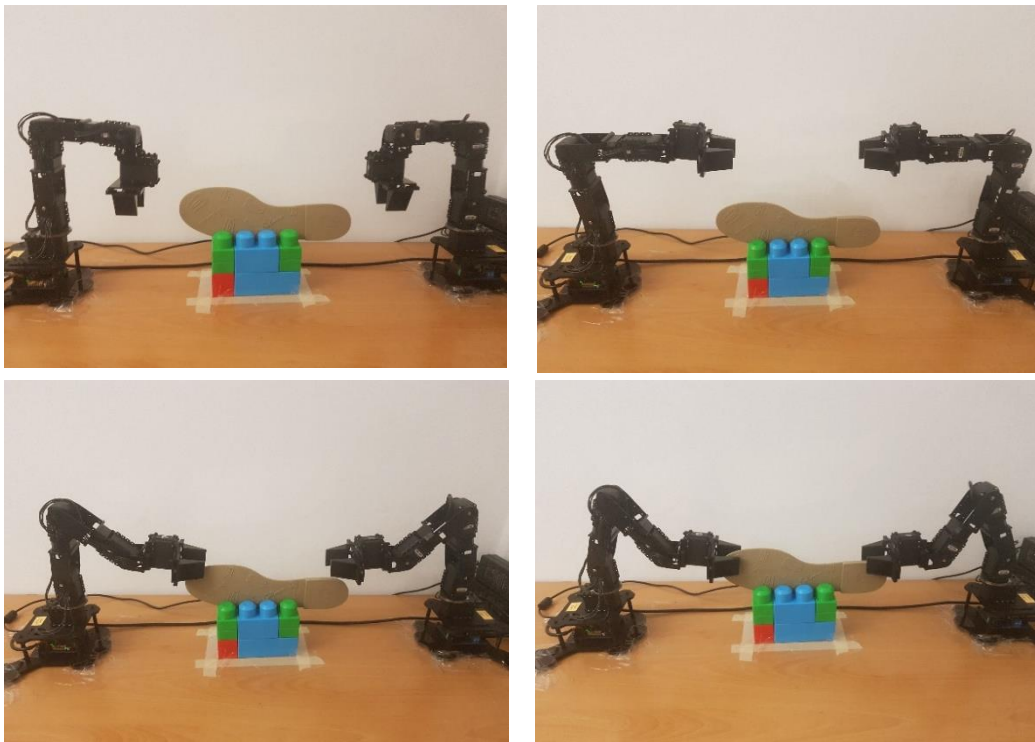


Figura 42 Secuencia de movimientos de la aplicación, parte I (orden de izquierda a derecha y de arriba abajo)

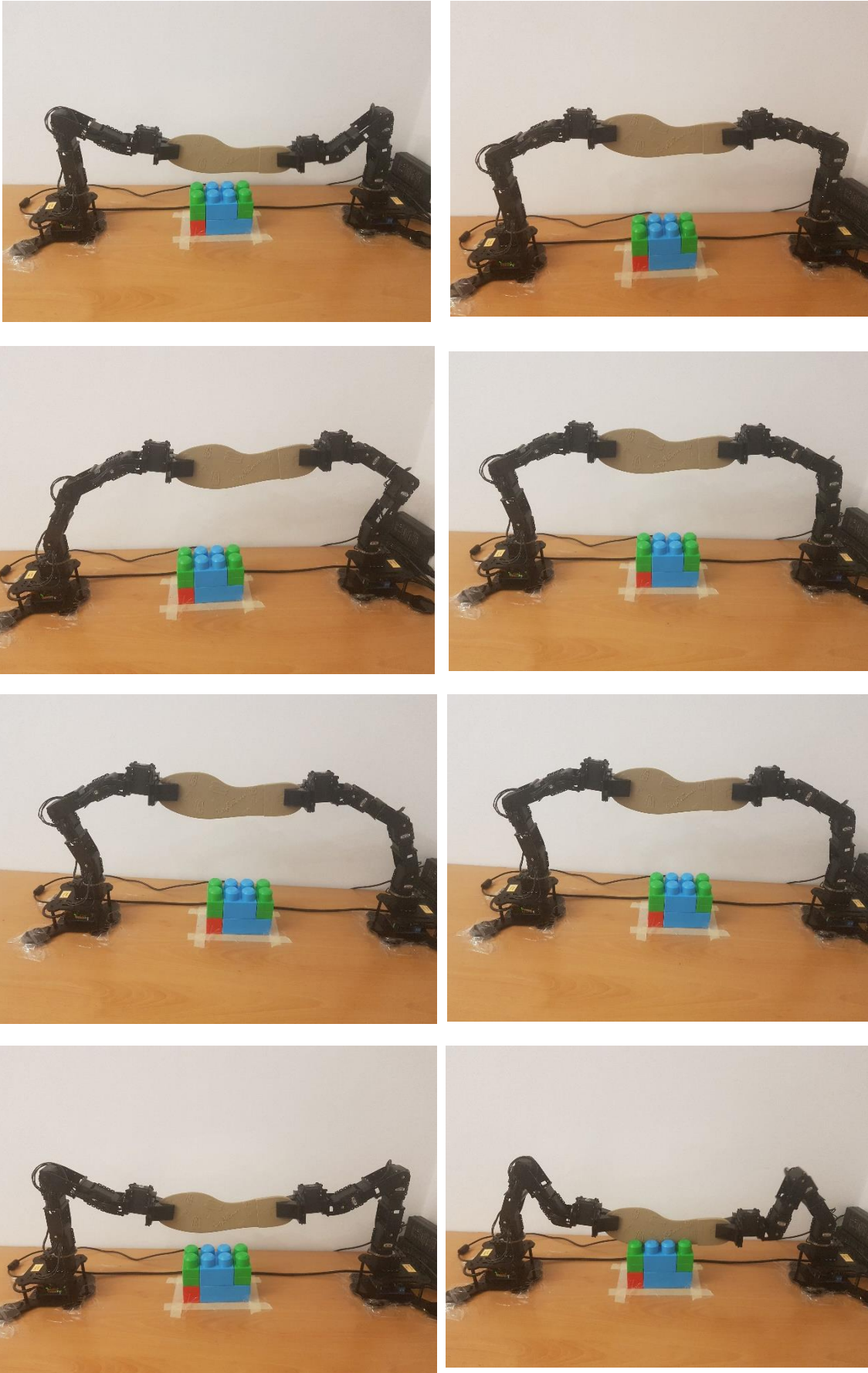


Figura 43 Secuencia de movimientos de la aplicación, parte II (orden de izquierda a derecha y de arriba abajo)

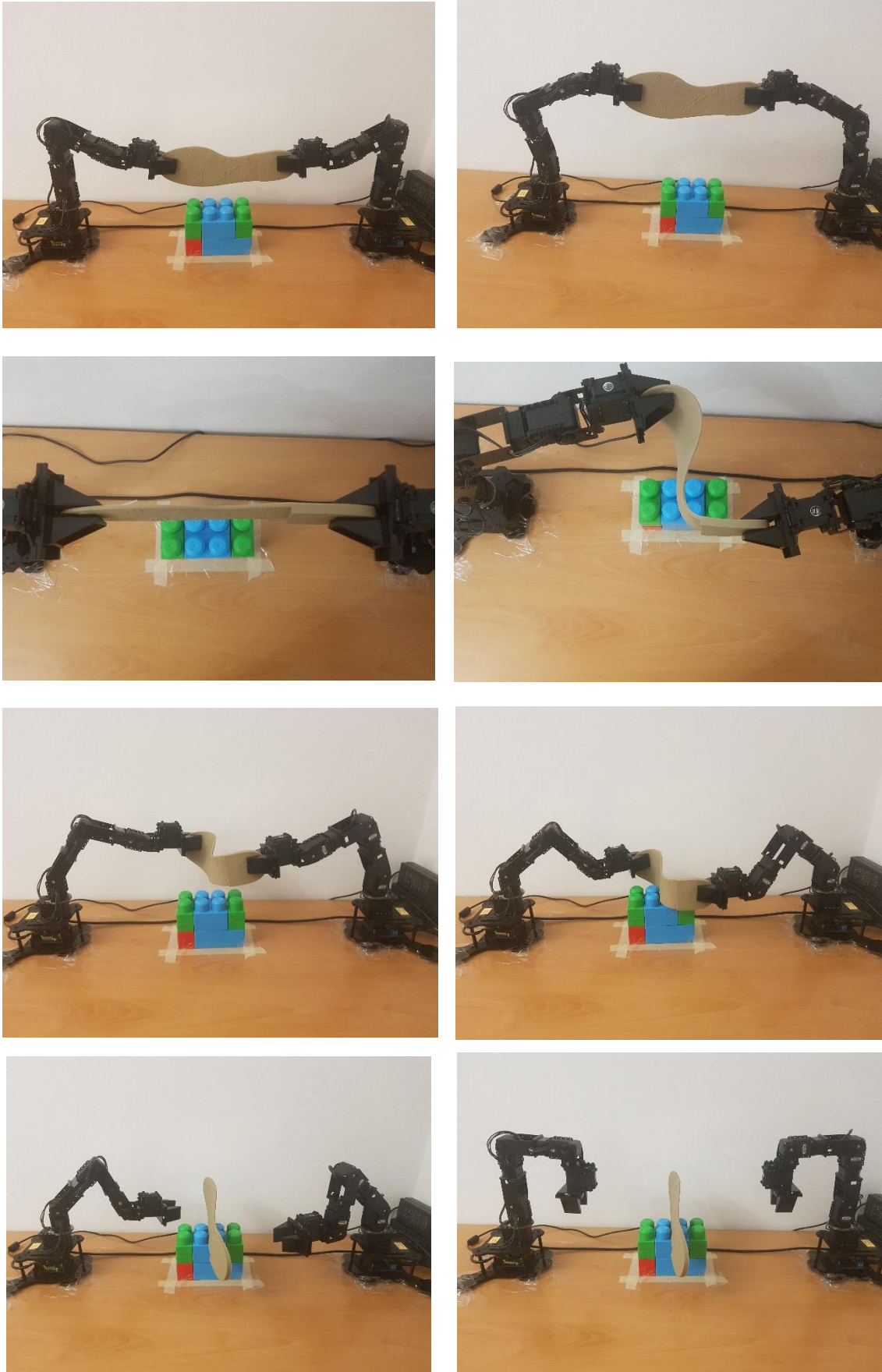


Figura 44 Secuencia de movimientos de la aplicación, parte III (orden de izquierda a derecha y de arriba abajo)

Como se puede observar en la figura 42, los robots parten del reposo a una posición que les permite descender en dirección del eje z y ponerse cerca de los extremos de la suela para a continuación cerrar la pinza.

En la figura 43 se observa que, una vez agarrada la plantilla, ambos robots se desplazan a la misma velocidad en dirección z hacia arriba. Desde ahí, se desplazan hacia la derecha y hacia la izquierda y vuelven de nuevo al centro. A continuación, el sistema multi-robot desciende para ubicar la suela en su posición original. Esto se realiza con el objetivo de demostrar que las trayectorias realizadas han sido bastante precisas, pues la plantilla no se ha descentrado de la posición original.

Por último, en la figura 44, sin soltar la suela, los robots vuelven a subir hacia arriba. Una vez ahí, de forma coordinada, los robots deforman la suela de tal forma que al bajar la plantilla encaje en los legos. Una vez encajada, los robots la sueltan, y vuelven a su posición de reposo.

5.4 Resultados y conclusiones

Finalmente se ha conseguido un movimiento coordinado entre ambos robots, y con una aplicación que requiere necesariamente el uso de un sistema multi-robot.

No obstante, el movimiento de dejar la suela sobre los legos, estando deformada, muchas de las veces, la plantilla se caía, porque las garras al abrirse y desplazarse lo arrastraban lo suficiente para que la suela se desequilibrara y cayera. Aproximadamente el 30% de las veces se conseguía terminar el movimiento.

Se observó que el error de posición aumentaba cuando ambos robots levantaban la suela. Este hecho es consecuencia, de lo que hemos comentado en el apartado 4.4, acerca de la falta de un controlador de la acción de par que no nos permita eliminar errores de posición. También se observó que en las trayectorias rectilíneas el robot iba paso a paso, de forma intermitente, a través de los puntos que hallamos mediante la cinemática inversa. Esto se debe, a que cada consigna que asignamos o escribimos al registro de *goal position*, las articulaciones desaceleran en cuanto detectan internamente que se están acercando a la consigna.

Otra de las dificultades que se encontró, fueron que la deformación de la suela tiende a separar las pinzas. Por ese motivo, la deformación que podíamos aplicarle se veía limitada.

6 Conclusiones

Abordar un movimiento coordinado, implicaba que los movimientos debían ser lo más precisos posibles, y se debía poder conocer la posición de los robots en cada instante para poder programarlos.

A pesar de ser brazos robóticos destinados al hobby, ofrecían una buena precisión, lo que permitió conseguir una buena coordinación. La mayoría de los fabricantes de robots para aficionados, incorporan librerías para trabajar en Arduino. Trabajar en un entorno diferente puede resultar una tarea complicada. Por ello, agradecemos al profesor australiano Peter Corke por el desarrollo de las librerías sobre las que nos hemos apoyado para trabajar en MatLab.

A partir de ahí, realizamos una aplicación de coordinación multi-robot, que consistía en rotar 90° una suela de calzado con dos robots de sólo 4 GDL. La aplicación que se ha mencionado en esta memoria ha sido una constante evolución de ideas primarias.

Antes de la aplicación, se realizaron pruebas en las que se evaluaba el funcionamiento de las funciones de las librerías. Se detectaron errores en las librerías, algunos relativamente fáciles de corregir mientras que otros implicaron adentrarse en código a bajo nivel, lo cual retrasó el trabajo, pero se aprendió mediante la práctica la interacción entre un ordenador y una placa.

Aunque no se tenía un controlador de par que nos permitiese eliminar perturbaciones debido a cargas, pudimos configurar parámetros dinámicos que modificaban la des/aceleración angular, con lo que se consiguió un movimiento más suave, pero, en detrimento de la precisión. No obstante, la mayor pérdida de precisión se debió a cargas externas, como la suela.

Para la aplicación, se recurrió al guiado manual para obtener los primeros puntos. A partir de ahí se aplicó la cinemática inversa para desarrollar trayectorias rectilíneas, aportando el código con los algoritmos desarrollados en la memoria.

A pesar de que la aplicación que hemos diseñado se podría hacer con un único robot de 6 GDL e incluso de 5 GDL; hacerla con robots de 4 GDL fue mucho más complejo, debido a que las orientaciones que podemos alcanzar son muy limitadas. Por otro lado, la deformación de la suela tendía a separar las pinzas. Afortunadamente nuestro robot resiste lo suficiente, frente a este tipo de suela que es muy fina.

No obstante, sólo el 30 % de las veces se conseguía dejar la suela sobre los legos, ya que el otro 70% de las veces, la deformación que experimentaba la suela junto al rozamiento entre la pinza y la plantilla en el momento en el que las garras se abrían, hacía que la suela se desplazara, se desequilibrara y cayera.

En conclusión, se han alcanzado los objetivos propuestos que consistían en el montaje y control coordinado de dos brazos robóticos. Y se ha añadido una aplicación especial que se planificó y programó, que consistía en girar una suela de calzado. El 30% de las veces se conseguía girar la plantilla. Por otro lado, no se observó un movimiento continuo en las trayectorias rectilíneas.

Como continuación, se podría seguir evaluando diferentes tipos de suelas que se diferencien en la forma, el peso y el material; y buscar alternativas para poder escribir consignas, de una trayectoria rectilínea, que estando muy cerca entre sí, el movimiento sea continuo y no intermitente.

7 Bibliografía

- [1] J. F. Engelberger, *Robotics in Practice: Management and applications of industrial robots*, 1980.
- [2] «UNIVERSAL-ROBOTS,» [En línea]. Available: https://www.universal-robots.com/es/wearecobots_congreso_2019/.
- [3] T. Robotics, «ArbotiX-M Robocontroller Getting Started Guide,» [En línea]. Available: <https://learn.trossenrobotics.com/index.php/getting-started-with-the-arbotix/7-arbotix-quick-start-guide>.
- [4] P. Corke, «Resources for robotics education: code, books and MOOCS,» [En línea]. Available: <https://petercorke.com/wordpress/interfacing-a-hobby-robot-arm-to-matlab>.
- [5] T. Robotics, «PhantomX Pincher Robot Arm Kit Mark II - Turtlebot Arm,» [En línea]. Available: <https://www.trossenrobotics.com/p/PhantomX-Pincher-Robot-Arm.aspx>.
- [6] R. e-manual, «ROBOTIS e-manual,» [En línea]. Available: <http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/#specifications>.
- [7] T. Robotics, «PhantomX Parallel Gripper Assembly Guide,» [En línea]. Available: <https://learn.trossenrobotics.com/projects/166-phantomx-parallel-gripper-assembly-guide.html>.
- [8] T. Robotics, «PhantomX Pincher Robot Arm Assembly Guide,» [En línea]. Available: <https://learn.trossenrobotics.com/16-interbotix/robot-arms/pincher-robot-arm/163-phantomx-pincher-robot-arm-assembly-guide.html>.
- [9] Arduino, «Descarga de Arduino versión 1.0.6,» [En línea].
- [10] T. Robotics, «PhantomX Pincher Robot Arm Build Check,» [En línea]. Available: <https://learn.trossenrobotics.com/interbotix/robot-arms/16-phantomx-pincher-robot-arm/25-phantomx-pincher-robot-arm-build-check>.
- [11] J. J. G. Campo, *El Problema Geométrico Inverso*, Zaragoza, 1999, pp. 2; 9-17.
- [12] A. Barrientos, L. F. Peñín, C. Balaguer y R. Aracil, *Fundamentos de Robótica*, Madrid: Mc Graw Hill, 1997, pp. 84-91.
- [13] P. Corke, *Robotics, Vision and Control*, Australia: Springer, 2017, p. 40.
- [14] A. Barrientos, M. Álvarez, J. Hernández, J. del Cerro y C. Rossi, «Modelado de Cadenas cinemáticas mediante Matrices de Desplazamiento. Una alternativa al método de Denavit-Hartenberg,» *Revista Iberoamericana de Automática e Informática Industrial*, nº 9, pp. 371-372, 2012.
- [15] K. M. Linch y F. C. Park, «Video Supplements for Modern Robotics,» Northwestern; Cambridge University Press, [En línea]. Available:

- <https://modernrobotics.northwestern.edu/nu-gm-book-resource/3-3-1-homogeneous-transformation-matrices/#department>.
- [16] J. J. Guerrero Campo, «Tema 5: Localización Espacial,» de *Apuntes de clase, Departamento de Automatización Flexible y Robótica de la Universidad de Zaragoza*, 2019.
- [17] J. J. Guerrero Campo, «Tema 6: Modelo Geométrico Directo,» de *Apuntes de clase. Departamento de Automatización Flexible y Robótica de la Universidad de Zaragoza*, 2019.
- [18] P. G. Mikell, W. Mitchell, N. N. Roger y G. O. Nicholas, *Robótica Industrial, Tecnología, Programación y Aplicaciones*, Mc Graw Hill, 1989.
- [19] R. B. a. G. McMurray, *Desarrollo algoritmico para matlab de la cinemática inversa del robot Puma 560 (6 GDL)*, *Georgia Institute of Technology*.
- [20] R. e-manual, «ROBOTIS e-manual,» [En línea]. Available: <http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/#cwccw-angle-limit6-8>.
- [21] R. e-manual, «ROBOTIS e-manual,» [En línea]. Available: <http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/>.
- [22] C. H, «robosavvy,» [En línea]. Available: <http://robosavvy.com/Builders/Chris.H/AX12Motor.pdf>.
- [23] E. Tira-Thompson, «Carnegie Mellon University, The Robotics Institute,» [En línea]. Available: https://www.ri.cmu.edu/pub_files/2009/3/CMU-RI-TR-09-41.pdf.
- [24] e.-m. ROBOTIS, «ROBOTIS e-manual,» [En línea]. Available: <http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/#control-table>.
- [25] R. e-manual, «Protocol 1.0 Communication overview,» [En línea]. Available: <http://emanual.robotis.com/docs/en/dxl/protocol1/#status-packet>.
- [26] T. Robotics, «AX12 MANUAL,» [En línea]. Available: [https://www.trossenrobotics.com/images/productdownloads/AX-12\(English\).pdf](https://www.trossenrobotics.com/images/productdownloads/AX-12(English).pdf).
- [27] D. Jacobson, «Introduction to Dynamixel Motor Control Using the ArbotiX-M Robocontroller, pp. 12;48,» [En línea]. Available: <http://biorobots.case.edu/wp-content/uploads/2014/12/IntroductiontoDynamixelMotorControlUsingtheArbotiX20141112-1.pdf>.
- [28] Ó. Justinico, P. Cárdenas y J. S. Rodríguez, «Control gestual de robot de 4 GDL con sensor LeapMotion».
- [29] e.-m. ROBOTIS, «ROBOTIS e-manual,» [En línea]. Available: <http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/#cw-compliance-margin>.
- [30] e.-m. ROBOTIS, «ROBOTIS e-manual compliance slope,» [En línea]. Available: <http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/#compliance-slope-28-29>.

- [31] e.-m. ROBOTIS , «ROBOTIS e-manual, punch,» [En línea].
- [32] S. Zilong, Z. Gang, E. Denis y P. Wilfrid, «Modelling and control of actuators with built-in position controller,» 2015.
- [33] e.-m. ROBOTIS, «ROBOTIS, e-manual Moving speed,» [En línea].
- [34] e.-m. ROBOTIS, «ROBOTIS e-manual, Goal position,» [En línea].
- [35] e.-m. ROBOTIS, «ROBOTIS e-manual, Present speed,» [En línea].
- [36] T. Robotics, «Setting DYNAMIXEL IDs with the DynaManager,» [En línea]. Available: <https://learn.trossenrobotics.com/index.php/getting-started-with-the-arbotix/1-using-the-tr-dynamixel-servo-tool#&panel1-2>.
- [37] TurtleBot. [En línea]. Available: <https://www.turtlebot.com/>.
- [38] R. P. Paul, Robot manipulators: mathematics, programming, and control: the computer control of robot manipulators, 1981, p. 42.
- [39] H. Toquica Cáceres. [En línea]. Available: https://www.researchgate.net/profile/Hans_Toquica_Caceres/publication/322222351_PhantomX_Pincher_Specifications/links/5a4d625aa6fdcc3e99d1616c/PhantomX-Pincher-Specifications.pdf.
- [40] e.-m. ROBOTIS, «ROBOTIS e-manual, Present Load,» [En línea].

8 Índices

8.2 Índice de figuras

Figura 1 PhantomX Pincher	1
Figura 2 Tipos de movimiento de cada una de las articulaciones del PhantomX Pincher [5].....	4
Figura 3 Partes mecánicas del robot	4
Figura 4 Placa ArbotiX-M, adaptador AC-DC y cable FTDI 5.0.....	5
Figura 5 Motores Dynamixel AX-12A.....	5
Figura 6 Distintas plataformas TurtleBots [37]	5
Figura 7 Kit de herramientas	6
Figura 8 Llaves necesarias	6
Figura 9 Motor: posición inicial incorrecta	6
Figura 10 Fotos del montaje del robot	7
Figura 11 Fotos de la prueba realizada para comprobar su correcto montaje. Movimiento de la segunda, tercera y cuarta articulación (de izquierda a derecha).....	8
Figura 12 Descripción gráfica de localización (posición + orientación) [38].....	10
Figura 13 Sistemas de referencias según DH sobre el robot en reposo	11
Figura 14 Modelado en MatLab del robot, en reposo.	12
Figura 15 Espacio de trabajo ampliado [39].....	13
Figura 16 Esquema de las referencias sobre las que trabajaremos	14
Figura 17 Limites articulares, Joint mode [21].....	16
Figura 18 Vista frontal del motor	16
Figura 19 Posición inicial (reposo) de los motores 1,2 y 5	17
Figura 20 Posición inicial (reposo) de los motores 3 y 4.	17
Figura 21 $\beta \in [-\pi, \pi]$	19
Figura 22 Respuesta ante una ejecución correcta de inicializacion_comunicacion.m	24
Figura 23 Configuración $q_1=[512\ 512\ 512\ 512\ 512]$ (izquierda) y configuración $q_2=[600\ 600\ 600\ 600\ 600]$ (derecha)	25
Figura 24 Código para alcanzar la posición q_2 a una velocidad v_1	25
Figura 25 Parámetros de control dinámico en un Dynamixel AX-12 [30].....	26
Figura 26 Parte del código <code>arb.setpos(varargin)</code> , versión original de Peter Corke	28
Figura 27 Parte del código <code>arb.setpos(varargin)</code> , versión corregida	29
Figura 28 Error en la lectura de una dirección del registro de control	30
Figura 29 Código para establecer la comunicación en modo depuración	30
Figura 30 Error en la lectura del paquete de datos recibido (Modo depuración).....	30
Figura 31 Máquina de estados para decodificar el paquete de estado, versión original (izquierda) y versión modificada (derecha).....	31
Figura 32 Modos de configuración del compliance slope [30]	33
Figura 33 Valor de varios registros de ambos robots.	33

<i>Figura 34 Posición 1</i>	36
<i>Figura 35 Posición 2</i>	36
<i>Figura 36 Incapacidad de giro en el eje z, e incapacidad de sujetar la suela.</i>	36
<i>Figura 37 Posición 3</i>	37
<i>Figura 38 Posición 4</i>	37
<i>Figura 39 Puntos de paso de una trayectoria rectilínea expresados en radianes (izquierda) y en marcas (derecha).</i>	39
<i>Figura 40 Simulación de trayectoria rectilínea sobre MatLab.</i>	40
<i>Figura 41 Movimiento desde la posición 3 (izquierda) a la posición 4 (derecha)</i>	41
<i>Figura 42 Secuencia de movimientos de la aplicación, parte I (orden de izquierda a derecha y de arriba abajo)</i>	41
<i>Figura 43 Secuencia de movimientos de la aplicación, parte II (orden de izquierda a derecha y de arriba abajo)</i>	42
<i>Figura 44 Secuencia de movimientos de la aplicación, parte III (orden de izquierda a derecha y de arriba abajo)</i>	43

8.3 Índice de gráficas

<i>Gráfica 1 Capacidad de carga con respecto a la distancia horizontal</i>	5
<i>Gráfica 2 De radianes a marcas de motores 1, 2 y 5</i>	17
<i>Gráfica 3 De radianes a marcas de motores 3 y 4</i>	17
<i>Gráfica 4 De marcas a radianes de motores 1, 2 y 5</i>	18
<i>Gráfica 5 De marcas a radianes de motores 3 y 4</i>	18
<i>Gráfica 6 Evolución de posición y velocidad en un primer tipo de experimento sobre un único robot.</i>	27
<i>Gráfica 7 Evolución de la posición de ambos robots moviéndose con diferente compliance_slope.</i>	34
<i>Gráfica 8 Evolución de la velocidad de ambos robots moviéndose con diferente compliance_slope.</i>	35

8.4 Índice de tablas

<i>Tabla 1 Capacidad de carga con respecto a la distancia horizontal</i>	5
<i>Tabla 2 Parámetros DH de nuestro robot en reposo.</i>	12

A. Anexos

A.1 Implementación del modelo geométrico en MatLab

Instalando la Robotics Toolbox

Como se ha comentado en la memoria, trabajaremos con la *Robotics Toolbox* de Corke debido a que se ha trabajado con ella, concretamente con la versión 9.8. Esta versión se puede descargar en: <http://www.petercorke.com/RTB/>.

Una vez descargado se lleva a la carpeta: Archivos de programa → MATLAB → R2018a → toolbox.

Las versiones antiguas, como ésta, requieren que cada vez que queramos hacer uso de la *Robotics Toolbox*, tengamos que abrir la carpeta en MatLab y ejecutar **startup_rvc.m**.

La tarea anterior no es complicada, pero puede resultar monótona hacerlo cada vez que abramos MatLab. Así que, investigando un poco acerca de las rutas de búsqueda, hemos visto que podemos cambiar la que viene por defecto, simplemente escribiendo el directorio donde se encuentra nuestro script o programa, y así poder ejecutarlo de manera más limpia y rápida:

```
addpath('C:\Program Files\MATLAB\R2018a\toolbox\rvctools');  
startup_rvc; %Se ejecuta el antiguo startup de Corke
```

Modelado geométrico en MatLab

A continuación, los parámetros 'd', 'a' y 'alpha' se introducen como variables de entrada en el objeto *Link*. Por defecto, reconoce que son los parámetros DH no modificados, también que el tipo de articulación es de rotación.

Link, no sólo permite introducir los parámetros de DH, sino también permite introducir variables dinámicas como la masa e inercias de los eslabones rígidos. Para más información, abrir el archivo *Link.m* que se encuentra en: Archivos de programa → MATLAB → R2018a → toolbox → rvctools → robot

Una vez introducido los parámetros de DH para cada uno de los objetos *Links*, debemos unirlos mediante *Seriallink*, que construye el objeto robot. Es aquí donde debemos introducir nuestra matriz de transformación homogénea entre la base 4 y la pinza, la matriz de paso entre la base mundo y la base del robot y el offset de las articulaciones 2 y 3

Todo lo anterior se encuentra en el programa **Phantom_X.m** (Ver Anexo 3). El dibujo del robot en la posición reposo que devuelve este programa se encuentra en la figura 13.

A.2 Desarrollo detallado del método de variables intermedias.

Nuestro objetivo es conocer las variables articulares necesarias para alcanzar una localización, ${}^M T_O$ requerida por el usuario. Dado que el quinto motor es de abrir y cerrar pinza no se tendrá en cuenta en este cálculo, su valor dependerá de la fuerza de agarre necesario.

En la figura 12, podemos ver claramente las bases sobre las que trabajaremos. Aunque sólo realizaremos el cálculo para un robot, esto es aplicable para el segundo robot.

Datos de partida:

Una vez que coloquemos a uno de los robots y el objeto que queremos alcanzar, conocemos las siguientes transformaciones: ${}^M T_O$ y ${}^M T_{B0}$.

Obtención de ${}^{B0} T_{B4}$:

Esta matriz contiene los valores numéricos de la posición y orientación de la base 4, y viene dada en función de los datos de partida y de la matriz de herramienta o pinza.

1) Hallamos ${}^{B0} T_O$:

$${}^M T_O = {}^M T_{B0} \cdot {}^{B0} T_O \rightarrow {}^{B0} T_O = ({}^M T_{B0})^{-1} \cdot {}^M T_O$$

2) Igualamos ${}^{B0} T_O$ a ${}^{B0} T_P$:

$$\left. \begin{array}{l} {}^{B0} T_P = {}^{B0} T_{B4} \cdot {}^{B4} T_P \\ {}^{B0} T_O = ({}^M T_{B0})^{-1} \cdot {}^M T_O \end{array} \right\} \begin{array}{l} {}^{B0} T_{B4} \cdot {}^{B4} T_P = ({}^M T_{B0})^{-1} \cdot {}^M T_O \\ \boxed{{}^{B0} T_{B4} = ({}^M T_{B0})^{-1} \cdot {}^M T_O \cdot ({}^{B4} T_P)^{-1}} \end{array}$$

Manipulación directa

El enfoque inicial sería igualar la matriz de paso, ${}^0 T_4$, y la matriz ${}^{B0} T_{B4}$.

$${}^0 T_4(\theta_1, \theta_2, \theta_3, \theta_4) = {}^{B0} T_{B4}$$

Obteniendo un sistema de 12 ecuaciones no lineales, dependientes y cuatro incógnitas, lo que dificulta su resolución analítica y el tiempo de cómputo aumenta.

Método de variables intermedias

Así que recurrimos a un método heurístico en el que se intenta obtener ecuaciones sencillas con una única dependencia articular.

- 1) Obtención de las matrices **U**, sustituyendo por variables auxiliares la matriz U_n si al construir la matriz U_{n-1} , la variable articular θ_{n-1} no es paralela a θ_n . Nunca se sustituirá por variables auxiliares la matriz $U_{n_{max}}$, en nuestro caso la U_4 .

Además, si una matriz U contiene variables articulares que forman ejes paralelos, se simplificarán dichas matrices por el ángulo suma.

La obtención de estas matrices manualmente puede resultar tedioso e incluso se puede llegar a cometer errores, por ello, se usa una manipulación de variables simbólicas en MatLab, tal como se puede ver en el script **matrices_de_paso.m** (Ver Anexo 3). Es importante tener definida la función **syndha.m** (Ver Anexo 3).

$$U_4 = {}^3T_4 = {}^3T_4(\theta_4)$$

$$U_3 = {}^2T_3 \cdot U_4 = {}^2T_3(\theta_3) \cdot U_4(\theta_4)$$

$$U_2 = {}^1T_2 \cdot U_3 = {}^1T_2(\theta_2) \cdot U_3(\theta_3, \theta_4); \text{ ejes } 2//3//4 \rightarrow \text{ No se sustituye } U_3 \text{ por variables intermedias}$$

$$U_1 = {}^0T_1 \cdot U_2 = {}^0T_4 = {}^0T_1(\theta_1) \cdot U_2(\theta_2, \theta_3, \theta_4); \text{ ejes } 1 \nparallel (2//3//4) \rightarrow \text{ se simplifica } U_2 \text{ por el ángulo suma y se sustituye por variables intermedias.}$$

$$U_2 = \begin{pmatrix} C_{234} & 0 & -S_{234} & a_3C_{23} + a_2C_2 \\ S_{234} & 0 & C_{234} & a_3S_{23} + a_2S_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} U_{211} & 0 & U_{213} & U_{214} \\ U_{221} & 0 & U_{223} & U_{224} \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Donde:

$$\begin{aligned} -C_{234} &= \cos(\theta_2 + \theta_3 + \theta_4) \\ -S_{234} &= \sin(\theta_2 + \theta_3 + \theta_4) \end{aligned}$$

- 2) Definimos las matrices **V**. Nos sirven para reducir la cadena cinemática cada vez que hallamos una variable articular. De esta forma, tenemos variables numéricas para conformar las ecuaciones con las matrices **U**.

Dado que sólo usaremos alguna de las componentes de cada una de las matrices **V**, es ineficiente calcularlas enteras, por lo que obtendremos simbólicamente dichas matrices a través del script **matrices_de_paso.m** (Ver Anexo 3).

La definición de las matrices **V** es la siguiente:

$$\begin{aligned} V_0 &= {}^{B0}T_{B4} \\ V_1 &= ({}^0T_1)^{-1} \cdot V_0 \\ V_2 &= ({}^1T_2)^{-1} \cdot V_1 \\ V_3 &= ({}^2T_3)^{-1} \cdot V_2 \end{aligned}$$

Donde la matriz, ${}^{B0}T_{B4}$, cuyos valores numéricos los conocemos, los sustituimos por valores simbólicos:

$${}^{B_0}T_{B_4} = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Las matrices V_1, V_2, V_3 contienen elementos que implican sumas y restas de senos y cosenos. Muchas de las veces son largos, por lo que para evitar su transcripción las sustituiremos simbólicamente. Desharemos el cambio cuando tengamos establecidas las ecuaciones.

Esto puede resultar confuso, pero se podrá apreciar la utilidad de esto en el desarrollo detallado de ecuaciones que se hará a continuación:

3) Cálculo de θ_1 :

Igualemos U_1 a V_0 y nos fijaremos en aquellas igualdades que nos permitan despejar la variable articular θ_1 :

$$U_1 = \begin{pmatrix} U_{211}C_1 & S_1 & U_{213}C_1 & U_{214}C_1 \\ U_{221}S_1 & -C_1 & U_{223}S_1 & U_{214}S_1 \\ -U_{221} & 0 & -U_{223} & -U_{224} \\ 0 & 0 & 0 & 1 \end{pmatrix}; \quad V_0 = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Si seleccionamos los elementos U_{112} y U_{122} e igualamos a o_x y o_y , obtenemos:

$$\left. \begin{array}{l} \sin \theta_1 = o_x \\ -\cos \theta_1 = o_y \end{array} \right\} \theta_1 = \text{atan}\left(\frac{o_x}{-o_y}\right)$$

El inconveniente que tiene introducir en MatLab la función *atan* es que no tiene en cuenta el signo del numerador y denominador, por lo que ángulo devuelto puede a veces no corresponderse con el cuadrante correcto. Además, devuelve una solución en rango $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Esto implica que perdemos soluciones. Para evitarlo usaremos la función *atan2* que tiene un rango $(-\pi, \pi]$ y tiene en cuenta ambos argumentos.

$$\theta_1 = \text{atan2}(o_x, -o_y)$$

4) Cálculo de θ_2 :

Igualemos U_2 a V_1 .

$$U_2 = \begin{pmatrix} C_{234} & 0 & -S_{234} & a_3C_{23} + a_2C_2 \\ S_{234} & 0 & C_{234} & a_3S_{23} + a_2S_2 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \quad V_1 = \begin{pmatrix} V_{111} & V_{112} & V_{113} & V_{114} \\ V_{121} & V_{122} & V_{123} & V_{124} \\ V_{131} & V_{132} & V_{133} & V_{134} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

En este caso, no hay un componente de la matriz U_2 que sólo dependa de θ_2 . Pero, si igualamos los términos de la fila 1 y 2 de la columna 4 de ambas matrices:

$$\left. \begin{aligned} a_3 C_{23} + a_2 C_2 &= V_{114} \\ a_3 S_{23} + a_2 S_2 &= V_{124} \end{aligned} \right\} \quad \left. \begin{aligned} a_3 C_{23} &= V_{114} - a_2 C_2 \\ a_3 S_{23} &= V_{124} - a_2 S_2 \end{aligned} \right\}$$

Elevamos al cuadrado ambos términos de las dos ecuaciones y las sumamos:

$$(a_3 C_{23})^2 + (a_3 S_{23})^2 = (V_{114} - a_2 C_2)^2 + (V_{124} - a_2 S_2)^2$$

Desarrollamos el lado izquierdo de la ecuación:

$$(a_3 C_{23})^2 + (a_3 S_{23})^2 = a_3^2 (C_{23}^2 + S_{23}^2) = a_3^2$$

Desarrollamos el lado derecho de la ecuación:

$$\begin{aligned} &(V_{114} - a_2 C_2)^2 + (V_{124} - a_2 S_2)^2 \\ &= V_{114}^2 - 2V_{114}a_2 C_2 + a_2^2 C_2^2 + V_{124}^2 - 2V_{124}a_2 S_2 + a_2^2 S_2^2 \end{aligned}$$

Igualando ambos miembros y despejando S_2 y C_2 :

$$a_3^2 = V_{114}^2 - 2V_{114}a_2 C_2 + a_2^2 C_2^2 + V_{124}^2 - 2V_{124}a_2 S_2 + a_2^2 S_2^2$$

$$a_3^2 = V_{114}^2 - 2V_{114}a_2 C_2 + V_{124}^2 - 2V_{124}a_2 S_2 + a_2^2 (C_2^2 + S_2^2)$$

$$a_3^2 = V_{114}^2 - 2V_{114}a_2 C_2 + V_{124}^2 - 2V_{124}a_2 S_2 + a_2^2$$

$$a_3^2 - a_2^2 - V_{114}^2 - V_{124}^2 = -2V_{114}a_2 C_2 - 2V_{124}a_2 S_2$$

$$a_3^2 - a_2^2 - V_{114}^2 - V_{124}^2 = -2a_2 (V_{114}C_2 - V_{124}S_2)$$

$$\frac{a_3^2 - a_2^2 - V_{114}^2 - V_{124}^2}{-2a_2} = V_{114}C_2 - V_{124}S_2$$

Nos queda una ecuación típica de la forma:

$$a \cos(\theta_i) + b \sin(\theta_i) = c$$

Donde existe solución si, $a^2 + b^2 \geq c^2$

$$2 \text{ soluciones: } \theta_i = \text{atan2}(b, a) \pm \arccos\left(\frac{c}{+\sqrt{a^2 + b^2}}\right)$$

En nuestro caso:

$$\theta_2 = \text{atan2}(V_{124}, V_{114}) \pm \arccos\left(\frac{\gamma}{+\sqrt{V_{114}^2 + V_{124}^2}}\right)$$

Donde:

$$\gamma = \frac{a_3^2 - a_2^2 - V_{114}^2 - V_{124}^2}{-2a_2}$$

Deshacemos el cambio de variable simbólico efectuado sobre la matriz V_1 :

$$V_{114} = p_x \cos(\theta_1) + p_y \sin(\theta_1)$$

$$V_{124} = -p_z$$

Como se puede observar, para la articulación 2 existe 2 posibles soluciones, en función de la solución que escojamos,

5) Cálculo de θ_3 :

Igualemos U_3 a V_2 .

$$U_3 = \begin{pmatrix} C_{34} & 0 & -S_{34} & a_3 C_3 \\ S_{34} & 0 & C_{34} & a_3 S_3 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \quad V_2 = \begin{pmatrix} V_{211} & V_{212} & V_{213} & V_{214} \\ V_{221} & V_{222} & V_{223} & V_{224} \\ V_{231} & V_{232} & V_{233} & V_{234} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Igualemos los términos de las filas 1 y 2 de la columna 4:

$$\left. \begin{array}{l} a_3 C_3 = V_{214} \\ a_3 S_3 = V_{224} \end{array} \right\} \begin{array}{l} \cos(\theta_3) = \frac{V_{214}}{a_3} \\ \sin(\theta_3) = \frac{V_{224}}{a_3} \end{array} \right\} \boxed{\theta_3 = \text{atan2}\left(\frac{V_{224}}{a_3}, \frac{V_{214}}{a_3}\right)}$$

Deshacemos el cambio de variable de V_2 :

$$V_{224} = V_{124}C_2 - V_{114}S_2 = -p_z \cos(\theta_2) - (p_x \cos(\theta_1) + p_y \sin(\theta_1)) \sin(\theta_2)$$

$$\begin{aligned} V_{214} &= V_{114}C_2 - a_2 + V_{124}S_2 \\ &= (p_x \cos(\theta_1) + p_y \sin(\theta_1)) \cos(\theta_2) - a_2 + (-p_z) \sin(\theta_2) \end{aligned}$$

6) Cálculo de θ_4 :

Igualemos U_4 a V_3 .

$$U_4 = \begin{pmatrix} C_4 & 0 & -S_4 & 0 \\ S_4 & 0 & C_4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}; \quad V_3 = \begin{pmatrix} V_{311} & V_{312} & V_{313} & V_{314} \\ V_{321} & V_{322} & V_{323} & V_{324} \\ V_{331} & V_{332} & V_{333} & V_{334} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Iguualamos las filas 1 y 2 de la columna 1:

$$\left. \begin{array}{l} C_4 = V_{311} \\ S_4 = V_{321} \end{array} \right\} \boxed{\theta_4 = \text{atan2}(V_{321}, V_{311})}$$

Deshacemos el cambio de variable de V_3 :

$$V_{321} = V_{221}C_3 - V_{211}S_3 = (V_{121}C_2 - V_{111}S_2)C_3 - (V_{111}C_2 + V_{121}S_2)S_3$$

$$V_{311} = V_{211}C_3 + V_{221}S_3 = (V_{111}C_2 + V_{121}S_2)C_3 + (V_{121}C_2 - V_{111}S_2)S_3$$

Donde:

$$V_{121} = -n_z$$

$$V_{111} = n_x \cos(\theta_1) + n_y \sin(\theta_1)$$

A.3 Programas y funciones de MatLab

Phantom_X.m

```
%%% Phantom_X.m
%Introduce los parámetros de DH de nuestro robot Phantom_X y construye el
%objeto PhantomX.
%
%En él se definen también la matriz de paso entre la base de la muñeca y la
%pinza y la matriz de paso entre la base mundo y la base del robot.
%
%Autor: Josué David Poma, Diciembre 2018
%Universidad de Zaragoza

% Inicializando la Robotic Toolbox de 2011
addpath('C:\Program Files\MATLAB\R2018a\toolbox\rvctools');
startup_rvc; %Se ejecuta el antiguo startup de Cork

% Contruyendo el modelo geométrico del robot
L(1) = Link('d', 0, 'a', 0, 'alpha', -pi/2);
L(2) = Link('d', 0, 'a', -0.107, 'alpha', 0);
L(3) = Link('d', 0, 'a', 0.107, 'alpha', 0);
L(4) = Link('d', 0, 'a', 0, 'alpha', -pi/2);

Tool= [1 0 0 0; 0 1 0 0; 0 0 1 0.109; 0 0 0 1]; %Matriz Homog. herramienta
Offset = [0 pi/2 -pi/2 0];
World = eye(4); %A modificar cuando realizemos las aplicaciones.

PhantomX = SerialLink(L, 'name',
'PhantomX', 'tool', Tool, 'offset', Offset, 'base', World);

q = [0 0 0 0]; %Posición de reposo
PhantomX.plot(q); %Dibujamos el robot en la posición de reposo
PhantomX.teach(); %Abrimos el cuadro de mandos
```


Matrices_de_paso.m

```
%%MATRICES DE PASO
% Obtención simbólica de las matrices T, U y V para resolver el modelo
% geométrico inverso de robots manipuladores.
%
% Autor: Josué David Poma, Febrero 2019
% Versión adaptada a la de Josechu Guerrero, Diciembre 2014. Recurso
para el curso de robótica.
% Adaptación a toolbox v9.7 (Corke) - GLN Octubre 2013
%
%%Carga el modelo del Phantom_X
Phantom_X; % solo se necesitan los parámetros cinemáticos

% Extrae la matriz de los parámetros DH del modelo
dhPhantomX=[];
for i=1:4,
    link=PhantomX.links(i);
    if isempty(link.theta), t=0; else t=link.theta; end
    if isempty(link.d), d=0; else d=link.d; end
    if isempty(link.a), a=0; else a=link.a; end
    if isempty(link.alpha), b=0; else b=link.alpha; end
    dhPhantomX=[dhPhantomX; t d a b link.sigma ]; %[t d a b=alpha
                                                    tipo]

end;

% Define de forma simbólica las matrices de paso;
T01=symdha(dhPhantomX(:,1:5),1);
T12=symdha(dhPhantomX(:,1:5),2);
T23=symdha(dhPhantomX(:,1:5),3);
T34=symdha(dhPhantomX(:,1:5),4);

% Define las matrices compuestas;
U4=T34;
U3=T23*U4;
U2=T12*U3;
U1=T01*U2;

%U4 = f(t4)--> No genero variables auxiliares
%U3 = f(t3,t4)--> t2//t3//t4 No genero variables auxiliares
%U2 = f(t2,t3,t4)--> t1 no es paralelo a t2, por lo que genero variables
auxiliares para U2. Además, t2//t3//t4, por lo que primero
simplificaremos las expresiones de U2 con el ángulo suma

U2 = simplify(U2);

%Miramos U2 y se observa que quedan términos que dependen de más de una
variable, por lo que sustituimos por variables auxiliares.

U2(1,1)='U211';
U2(1,3)='U213';
U2(1,4)='U214';
U2(2,1)='U221';
U2(2,3)='U223';
U2(2,4)='U224';
```

```
%Obtenemos la nueva U1 como producto de T01 y U2 simplificada
U1=T01*U2
```

```
%Definición de las matrices V.
```

```
V0=str2sym('[nx, ox, ax, px; ny, oy, ay, py; nz, oz, az, pz; 0, 0, 0, 1]');
```

```
V1=inv(T01)*V0;
```

```
%Para evitar errores a la hora de igualar elementos de las matrices "U" con elementos de las matrices "V", se hace uso de variables intermedias al igual que se hizo con las matrices U. De esa manera es más ordenado.
```

```
V1 = simplify(V1)
```

```
V1(1,1)='V111';
```

```
V1(1,2)='V112';
```

```
V1(1,3)='V113';
```

```
V1(1,4)='V114';
```

```
V1(2,1)='V121';
```

```
V1(2,2)='V122';
```

```
V1(2,3)='V123';
```

```
V1(2,4)='V124';
```

```
V1(3,1)='V131';
```

```
V1(3,2)='V132';
```

```
V1(3,3)='V133';
```

```
V1(3,4)='V134';
```

```
V2=inv(T12)*V1;
```

```
V2 = simplify(V2)
```

```
V2(1,1)='V211';
```

```
V2(1,2)='V212';
```

```
V2(1,3)='V213';
```

```
V2(1,4)='V214';
```

```
V2(2,1)='V221';
```

```
V2(2,2)='V222';
```

```
V2(2,3)='V223';
```

```
V2(2,4)='V224';
```

```
V3=inv(T23)*V2;
```

```
V3= simplify(V3)
```

```
V3(1,1)='V311';
```

```
V3(1,2)='V312';
```

```
V3(1,3)='V313';
```

```
V3(1,4)='V314';
```

```
V3(3,1)='V331';
```

```
V3(3,2)='V332';
```

```
V3(3,3)='V333';
```

```
V3(3,4)='V334';
```

Symdha.m (Función)

```
%Construye una matriz simbólica a partir de los parámetros DH
%Uso:
% A = symdha(dh,n)
%Devuelve una matriz homogénea simbólica desde el eslabón na-1 al eslabón
na a partir de los datos cinemáticos (matriz de parámetros dh),
utilizando la representación de la Robotics Toolbox.
%
% MODIFICACIONES VARIAS: Josechu Guerrero - Enero 2000, Sept2000
% Adaptación a toolbox v9.7 - GLN Octubre 2013
%v9.7 => DH=[theta d a alpha] =[t d a b tipo] (en toolbox
anterior:[b=alpha, A, t=theta, D, tipo]

function A=symdha (dh,n)

%b=alpha, t=theta in radianas
%create symbolic A matrix
a=sym('a');
b=sym('b');
t=sym('t');
d=sym('d');
Asym=[cos(t) -sin(t)*cos(b) sin(t)*sin(b) a*cos(t);sin(t) cos(t)*cos(b)
-cos(t)*sin(b) a*sin(t);0 sin(b) cos(b) d;0 0 0 1];
%Asym=sym(' [cos(t) -sin(t)*cos(b) sin(t)*sin(b) a*cos(t);sin(t)
cos(t)*cos(b) -cos(t)*sin(b) a*sin(t);0 sin(b) cos(b) d;0 0 0 1]');

var=str2sym('[a1 a2 a3 a4 a5 a6 a7 t1 t2 t3 t4 t5 t6 t7 d1 d2 d3 d4 d5
d6 d7]');

A=subs (Asym, 'b', sym(dh(n,4))); %subs in numerical twist angle of link

if dh(n,3)==0 %checks for existence of link length a
    A=subs (A, 'a', 0); %subs in zero for link length
else
    A=subs (A, 'a', var(n)); %subs in variable link length
end

if dh(n,5)==0 %checks whether link is revolute(0) or prismatic(non-0)
    A=subs (A, 't', var(7+n)); %subs in variable joint angle if revolute
    if dh(n,2)==0
        A=subs (A, 'd', 0); %subs in zero for joint offset length
    else
        A=subs (A, 'd', var(14+n)); %subs in variable joint offset length
    end
else
    A=subs (A, 'd', var(14+n)); %subs in variable joint offset length
if prismatic
    A=subs (A, 't', sym(dh(n,1))); %subs in the numerical joint
angle

end

A=simplify(A);
```

Ikine_PhantomX.m

```
%%% IKINEPHANTOMX
% Función que devuelve las variables articulares en función de la
% localización deseada que se quiera alcanzar. No se tiene en cuenta
% limitaciones articulares.
%
% Reference:
% - Inverse kinematics for a PUMA 560,
%   Paul and Zhang,
%   The International Journal of Robotics Research,
%   Vol. 5, No. 2, Summer 1986, p. 32-44
%
% Author::
% Robert Biro with Gary Von McMurray,
% GTRI/ATRP/IIMB,
% Georgia Institute of Technology
% 2/13/95
%
% See also SerialLink.FKINE, SerialLink.IKINE.
%
% Modificaciones varias para trabajo asignatura AFR
% Josechu Guerrero, ISA, Universidad de Zaragoza
% Octubre 2015
%
% Adaptación para nuestro robot PhantomX
% Josué David Poma, Febrero 2019
% Universidad de Zaragoza
function theta = ikinePhantomX(robot, T, varargin)
    % Se le pasa 3 parámetros a esta función
    %robot: nuestro objeto 'PhantomX'
    %T: matriz de localización que contiene la posición y
    %   orientación deseada
    %varargin: 'd' o 'u'; indica la configuración (codo arriba o
    %         codo abajo) que queremos debido a la doble solución.
    %         %ikinePhantomX(PhantomX, T, 'd') --> DOWN
    %         %ikinePhantomX(PhantomX, T, 'u') --> UP (Por defecto)

    % Captura los parametros de Denavit Hartenberg del robot necesarios
    % Distancias DH NO nulas del PhantomX
    L = robot.links;
    a2 = L(2).a;
    a3 = L(3).a;

    % En el caso de haber definido bases de referencia (mundo) y de la
    % pinza
    T = inv(robot.base) * T*inv(robot.tool);

    % Renombrar los elementos de la matriz homogenea destino segun la
    % forma T=[N O A P] , para facilitar su utilización en el código
    %posterior
    Nx = T(1,1);  Ox = T(1,2);  Ax = T(1,3);  Px = T(1,4);
    Ny = T(2,1);  Oy = T(2,2);  Ay = T(2,3);  Py = T(2,4);
    Nz = T(3,1);  Oz = T(3,2);  Az = T(3,3);  Pz = T(3,4);
```

```

%Como se verá más adelante hay dos configuraciones posibles para
%una posición:
%     - Codo arriba (UP)
%     - Codo abajo (DOWN)
%
% El tercer argumento ('varargin'):
%   ikinePhantomX(PhantomX, T, 'd') --> DOWN
%   ikinePhantomX(PhantomX, T, 'u') --> UP

if nargin < 1
    configuration = '';
else
    configuration = lower(varargin);
end

% Configuración por defecto
n1 = 1;    % Codo arriba

if contains(configuration, 'd')
    n1 = -1;
end
if contains(configuration, 'u')
    n1 = 1;
end

%% RESOLUCIÓN DEL MODELO GEOMETRICO-CINEMATICO INVERSO

% Hallando theta(1)
theta(1) = atan2(Ox, -Oy);

% Hallando theta(2)
V114 = Px*cos(theta(1)) + Py*sin(theta(1));
V124 = -Pz;
aux3 = (a3^2-a2^2-V114^2-V124^2)/(-2*a2);
aux4 = sqrt(V114^2+V124^2);

    if ~isreal(aux4)
        warning('punto no alcanzable');
        theta = [NaN NaN NaN NaN];
        return
    end
theta(2) = atan2(V124,V114)+n1*acos(aux3/aux4); %Doble
solución

% Hallando theta(3)

%aux5 = sin(theta3)
aux5 = (V124*cos(theta(2))-V114*sin(theta(2)))/a3;

%aux6 = cos(theta3)
aux6 = (V114*cos(theta(2)) - a2 + V124*sin(theta(2)))/a3;

theta(3) = atan2(aux5,aux6);

% Hallando theta(4)

```

```

V111 = Nx*cos(theta(1)) + Ny*sin(theta(1));
V121 = -Nz;

V211 = V111*cos(theta(2)) + V121*sin(theta(2));
V221 = V121*cos(theta(2)) - V111*sin(theta(2));

%V321 = sin(theta(4))
V321 = V221*cos(theta(3)) - V211*sin(theta(3));

%V311 = cos(theta(4))
V311 = V211*cos(theta(3)) + V221*sin(theta(3));

theta(4) = atan2(V321,V311);

% Para el caso de haber definido offset en las variables
articulares
for i=1:robot.n %#ok<*AGROW>
    theta(i) = theta(i) - L(i).offset;
end

%Rango [-pi,pi]
for i=1:4
    theta(i)=atan2(sin(theta(i)),cos(theta(i)));
end
%Para los motores 3 y 4

for i=3:4
    if theta(i)>(pi/3)
        theta(i)=-pi-(pi-theta(i));
    end
end
end

```

Inicializacion_comunicacion.m

```
%%IMPORTANTE:
% Antes de ejecutar este programa, es necesario cargar el programa
%pyypose.ino en la placa ARBOTIX-M desde la plataforma Arduino. Dicho
%programa se encuentra la librería de Arbotix, que se puede descargar
%en: https://learn.trossenrobotics.com/interbotix/robot-arms/16-phantomx-pincher-robot-arm/25-phantomx-pincher-robot-arm-build-check
%
% Porque en él se establece la comunicación entre la placa y los
%motores con una velocidad de 1 Mbps.
% Autor: Josué David Poma, junio 2019
% Universidad de Zaragoza

%% CONFIGURANDO COMUNICACIÓN ORDENADOR-->ARBOTIX-M
%% Inicializando la toolbox de 2018

addpath('C:\Program Files\MATLAB\R2018a\toolbox\rvctools_2');
startup_rvc; %Se ejecuta el nuevo startup de Corke, necesario
            %para poder ejecutar la función arbotix.

%% Inicializamos la comunicación entre matlab-> placa ARBOTIX-M (38400
baud)

addpath(fullfile( fileparts(which('startup_rvc')), 'robot',
'interfaces'));

%Necesario para que se ejecute la función Arbotix que se encuentra
%en la carpeta 'interfaces'.

robot1 = Arbotix('port', 'COM5', 'nservos', 5,'baud',38400) %Se
%establece la comunicación entre el ordenador y la placa ARBOTIX-M
%del robot nº1.

pause(8); %Necesario para que se ejecute la orden anterior

robot2 = Arbotix('port', 'COM4', 'nservos', 5,'baud',38400) %Se
%establece la comunicación entre el ordenador y la placa ARBOTIX-M
%del robot nº2.

pause(8); %Necesario para que se ejecute la orden anterior

%arb.disconnect()
```

Arbotix.m (Versión modificada)

```
%Arbotix Interface to Arbotix robot-arm controller
%
% A concrete subclass of the abstract Machine class that implements a
% connection over a serial port to an Arbotix robot-arm controller.
%
% Methods::
% Arbotix      Constructor, establishes serial communications
% delete      Destructor, closes serial connection
% getpos      Get joint angles
% setpos      Set joint angles and optionally speed
% setpath     Load a trajectory into Arbotix RAM
% relax       Control relax (zero torque) state
% settled     Control LEDs on servos
% gettemp     Temperature of motors
%
% getcwcompmar  Get CW compliance margin
% getccwcompmar Get CCW compliance margin
% getcwcompslope Get CW compliance slope
% getccwcompslope Get CCW compliance slope
% getpunch     Get punch
% getcurspeed  Get current speed
% getspeed    Get setting speed
%
% setcwmargin  Set CW compliance margin
% setccwmargin Set CCW compliance margin
% set0063wslope  Set CW compliance slope
% setccwslope  Set CCW compliance slope
%
% writedata1  Write byte data to servo control table
% writedata2  Write word data to servo control table
% readdata    Read servo control table
%-
% command     Execute command on servo
% flush       Flushes serial data buffer
% receive     Receive data
%
% Example::
%   arb=Arbotix('port', '/dev/tty.usbserial-A800JDPN',
% 'nservos', 5);
%   q = arb.getpos();
%   arb.setpos(q + 0.1);
%
% Notes::
% - This is experimental code.
% - Considers the robot as a string of motors, and the last joint is
%   assumed to be the gripper. This should be abstracted, at the
%   moment this
%   is done in RobotArm.
% - Connects via serial port to an Arbotix controller running the
%   pypose
%   sketch.
%
% See also Machine, RobotArm.

% Copyright (C) 1993-2015, by Peter I. Corke
% MODIFICACIONES: Josué David Poma, julio 2019 Universidad de Zaragoza
```



```

% This file is part of The Robotics Toolbox for MATLAB (RTB).
%
% RTB is free software: you can redistribute it and/or modify
% it under the terms of the GNU Lesser General Public License as
% published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% RTB is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU Lesser General Public License for more details.
%
% You should have received a copy of the GNU Lesser General Public
% License
% along with RTB. If not, see <http://www.gnu.org/licenses/>.
%
% http://www.petercorke.com

% Copyright (c) 2013 Peter Corke

% only a subset of Arbotix commands supported
% READDATA, WRITEDATA (1 or 2 bytes only)
% WRITESYNC and ACTION not supported but should be

% Should subclass an abstract superclass Machine

classdef Arbotix < Machine

    properties
        serPort;
        nservos;

        gripper
    end

    properties (Constant)
        %% define some important memory locations from EEPROM Area
        % This values are not volatiles
        ADDR_VERSION = 2;
        ADDR_DELAYTIME = 5;
        ADDR_CWLIMIT = 6;
        ADDR_CCWLIMIT = 8;
        ADDR_ALARM_LED = 17;
        ADDR_ALARM_SHUTDOWN = 18;

        %% Define memory locations from RAM Area
        % Always initial values will be dates of the tables INITIAL
        VALUES
        ADDR_TORQUE = 24; % 0(0x00)
        ADDR_LED = 25; % 0(0x00)
        ADDR_CW_COMPLIANCE_MARGIN = 26; % 1(0x01)
        ADDR_CCW_COMPLIANCE_MARGIN = 27; % 1(0x01)
        ADDR_CW_COMPLIANCE_SLOPE = 28; % 32(0x20)
        ADDR_CCW_COMPLIANCE_SLOPE = 29; % 32(0x20)
        ADDR_GOAL = 30; %Goal Position % [ADDR36]value
    end
end

```

```

ADDR_SPEED = 32; %Setting speed      % 0(0x00)
ADDR_POS = 36;  %Current position   % ¿?
ADDR_CURRENT_SPEED = 38;            % ¿?
ADDR_TEMP = 43;                      % ¿?
ADDR_PUNCH = 48;                     % 32(0x20)

%%% define the instructions
% native Arbotix instructions
INS_READ_DATA = 2;
INS_WRITE_DATA = 3;

% pseudo Arbotix instructions, implemented by Arbotix pypose
INS_ARB_SIZE_POSE = 7;
INS_ARB_LOAD_POSE = 8;
INS_ARB_LOAD_SEQ = 9;
INS_ARB_PLAY_SEQ = 10;
INS_ARB_LOOP_SEQ = 11;
INS_ARB_TEST = 25;
end

% Communications format:
%
% To Arbotix:
%
% 0xff 0xff ID  LEN  INSTRUC  PAYLOAD  CHKSUM
%
% ID single byte, identifies the servo on the bus 0-(N-1)
% LEN single byte, indicates the number of bytes to follow the LEN
% byte incl checksum
% INSTRUC a single byte saying what to do:
% 2  read data from control table: PAYLOAD=START_ADR, NUM_BYTES
% 3  write data to control table: PAYLOAD=START_ADR, P1, P2, ...
%
% 7  set #DOF: PAYLOAD=NP
% 8  load pose: PAYLOAD=POSE# $Q_1 $Q_2 ... $Q_NP
% 9  load sequence: PAYLOAD=POSE# $T POSE# $T ... 255
% 10 run sequence
% 11 loop sequence
%
% CHKSUM is a single byte, modulo 256 sum over ID .. PAYLOAD.
%
% Notes::
% - $X means that X is a 16-bit (word) quantity.
% - Arbotix only passes instructions 2 and 3 to the servos, other
%   Dynamixel instructions are inaccessible.
% - Instructions 7-11 are implemented by the Arbotix pypose
%   sketch.
%
% Methods::
% writedata1  Writes byte data to control table
% writedata2  Writes word data to control table
% readdata    Reads data from control table
% command     Invokes instruction on servo

methods
function arb = Arbotix(varargin)

```

```

%Arbotix.Arbotix Create Arbotix interface object
%
% ARB = Arbotix(OPTIONS) is an object that represents a
% connection to a chain of Arbotix servos connected via an
% Arbotix controller and serial link to the host computer.
%
% Options::
% 'port',P      Name of the serial port device,
%               eg. /dev/tty.USB0
% 'baud',B      Set baud rate (default 38400)
% 'debug',D     Debug level, show communications packets
%               (default 0)
% 'nservos',N   Number of servos in the chain

opt.port = '';
opt.debug = false;
opt.nservos = [];
opt.baud = 38400;

opt = tb_optparse(opt, varargin);

arb.serPort = opt.port;
arb.debug = opt.debug;
arb.nservos = opt.nservos;

arb.connect(opt);

% open and closed amount
arb.gripper = [0 2.6];

end

function connect(arb, opt)
%Arbotix.connect Connect to the physical robot controller
%
% ARB.connect() establish a serial connection to the
% physical robot controller.
%
% See also Arbotix.disconnect.

% clean up any previous instances of the port, can
% happen...
for tty = instrfind('port', opt.port)
    if ~isempty(tty)

        disp(['serPort ' tty.port 'is in use. Closing
              it.'])

        fclose(tty);
        delete(tty);
    end
end

if opt.verbose
    disp('Establishing connection to Arbotix chain...');
end

```

```

arb.serPort = serial(arb.serPort, 'BaudRate', opt.baud);
set(arb.serPort, 'InputBufferSize', 1000)
set(arb.serPort, 'Timeout', 1)
set(arb.serPort, 'Tag', 'Arbotix')

if opt.verbose
    disp('Opening connection to Arbotix chain...');
end

pause(0.5);
try
    fopen(arb.serPort);
catch me
    disp('open failed');
    me.message
    return
end

arb.flush();
end

function disconnect(arb)
    % Arbotix.disconnect Disconnect from the physical robot
    % controller
    %
    % ARB.disconnect() closes the serial connection.
    %
    % See also Arbotix.connect.

    tty = instrfind('port', arb.serPort.port);
    fclose(tty);
    delete(tty);
end

function s = char(arb)
    %Arbotix.char Convert Arbotix status to string
    %
    % C = ARB.char() is a string that succinctly describes
    % the status of the Arbotix controller link.

    % show serport Status, number of servos
    s = sprintf('Arbotix chain on serPort %s (%s)', ...
        arb.serPort.port, get(arb.serPort, 'Status'));
    if arb.nservos
        s = strvcats(s, sprintf(' %d servos in chain',
            arb.nservos));
    end
end

function display(arb)
    %Arbotix.display Display parameters
    %
    % ARB.display() displays the servo parameters in compact
    % single line format.
    %
    % Notes::

```

```

% - This method is invoked implicitly at the command line
% when the result of an expression is a Arbotix object and
% the command has no trailing semicolon.
%
% See also Arbotix.char.
loose = strcmp( get(0, 'FormatSpacing'), 'loose');
if loose
    disp(' ');
end
disp([inputname(1), ' = '])
disp( char(arb) );
end % display()

```

```

function p = getpos(arb, id)
%Arbotix.getpos Get position
%
% P = ARB.GETPOS(ID) is the position (0-1023) of servo ID.
%
% P = ARB.GETPOS([]) is a vector (1xN) of positions of
% servos 1 to N.
%
% Notes::
% - N is defined at construction time by the 'nservos'
% option.
%
% See also Arbotix.e2a.

arb.flush();

if nargin < 2
    id = [];
end

if ~isempty(id)
    retval = arb.readdata(id, Arbotix.ADDR_POS, 2);
    p = retval.params*[1; 256];
else
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end
    p = [];
    for j=1:arb.nservos
        retval = arb.readdata(j, Arbotix.ADDR_POS, 2);
        p(j) = retval.params*[1; 256];
    end
end
end
end

```

```

%% This code is a contribution from Josué David Poma 15/07/2019
%% Universidad de Zaragoza

```

```

function speed = getspeed (arb,id)
%It shows the setting angular speed in [0,1023]. 0 means
%largest speed.
arb.flush();

```

```

if nargin<2
    id=[];
end

if ~isempty(id)
    retval = arb.readdata(id,Arbotix.ADDR_SPEED,2);
    speed = retval.params*[1;256];
else
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end
    speed = [];
    for j=1:arb.nservos
        retval = arb.readdata(j,Arbotix.ADDR_SPEED,2);
        speed(j)=retval.params*[1;256];
    end
end
end

function current_speed = getcurspeed (arb,id)
%It shows the current angular speed in [0,1023]. 0 means
%largest speed.
arb.flush();
if nargin<2
    id=[];
end

if ~isempty(id)
    retval=arb.readdata(id,Arbotix.ADDR_CURRENT_SPEED,2);
    current_speed = retval.params*[1;256];
else
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end
    current_speed = [];
    for j=1:arb.nservos
        retval=arb.readdata(j,Arbotix.ADDR_CURRENT_SPEED,2);
        current_speed(j)=retval.params*[1;256];
    end
end
end

function cwcompar = getcwcompar (arb,id)
%It shows the CW (clock wise) compliance margin. It's like the
%error in the CW direction margin
arb.flush();
if nargin<2
    id=[];
end

if ~isempty(id)
    retval=arb.readdata(id,Arbotix.ADDR_CW_COMPLIANCE_MARGI
N,1);
    cwcompar = retval.params;
else

```

```

        if isempty(arb.nservos)
            error('RTB:Arbotix:notspec', 'Number of servos not
                specified');
        end
        cwcompmar = [];
        for j=1:arb.nservos

            retval=arb.readdata(j,Arbotix.ADDR_CW_COMPLIANCE_M
                ARGIN,1);
            cwcompmar(j)=retval.params;
        end
    end
end

function cwcompmar = getccwcompmar (arb,id)
%It shows the CCW (counter clock wise) compliance margin. It's
like the error in the CCW direction margin
    arb.flush();
    if nargin<2
        id=[];
    end

    if ~isempty(id)
        retval=arb.readdata(id,Arbotix.ADDR_CCW_COMPLIANCE_MAR
            GIN,1);
        cwcompmar = retval.params;
    else
        if isempty(arb.nservos)
            error('RTB:Arbotix:notspec', 'Number of servos not
                specified');
        end
        cwcompmar = [];
        for j=1:arb.nservos
            retval=arb.readdata(j,Arbotix.ADDR_CCW_COMPLIANCE_
                MARGIN,1);
            cwcompmar(j)=retval.params;
        end
    end
end

function cwcompslope = getcwcompslope (arb,id)
%It shows the CW (clock wise) compliance slope.
    arb.flush();
    if nargin<2
        id=[];
    end

    if ~isempty(id)
        retval=arb.readdata(id,Arbotix.ADDR_CW_COMPLIANCE_SLOP
            E,1);
        cwcompslope = retval.params;
    else
        if isempty(arb.nservos)
            error('RTB:Arbotix:notspec', 'Number of servos not
                specified');
        end
        cwcompslope = [];
        for j=1:arb.nservos

```

```

        retval=arb.readdata(j,Arbotix.ADDR_CW_COMPLIANCE_S
        LOPE,1);
        cwcompslope(j)=retval.params;
    end
end
end

function ccwcompslope = getccwcompslope (arb,id)
%It shows the CCW (counter clock wise) compliance slope.
arb.flush();
if nargin<2
    id=[];
end

if ~isempty(id)
    retval=arb.readdata(id,Arbotix.ADDR_CCW_COMPLIANCE_SLO
    PE,1);
    ccwcompslope = retval.params;
else
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
        specified');
    end
    ccwcompslope = [];
    for j=1:arb.nservos
        retval=arb.readdata(j,Arbotix.ADDR_CCW_COMPLIANCE_
        SLOPE,1);
        ccwcompslope(j)=retval.params;
    end
end
end

function punch = getpunch (arb,id)
%It shows punch. This parameter is useful for controlling
dynamic
arb.flush();
if nargin<2
    id=[];
end

if ~isempty(id)
    retval = arb.readdata(id,Arbotix.ADDR_PUNCH,2);
    punch = retval.params*[1;256];
else
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
        specified');
    end
    punch = [];
    for j=1:arb.nservos
        retval = arb.readdata(j,Arbotix.ADDR_PUNCH,2);
        punch(j)=retval.params*[1;256];
    end
end
end
end

```



```

function setcwmargin(arb,varargin)
    %Arbotix.setcwmargin
    %
    %ARB.SETCWMARGIN(ID,CWMARGIN) sets the cw compliance
    %margin [0,255] of servo ID

    %ARB.SETCWMARGIN(CWMARGIN) %sets the cw compliance margin
    %of servos 1-N to corresponding elements of the vector
    %POS(1xN).
    %
    %Notes:
    %- ID is in the range 1 to N
    %- N is defined at construction time by the 'nservos'
    %option
    %- CW compliance margin varies from 0 to 255. It means the
    %error between goal position and present position in the
    %clock wise direction. The unit is the same as Goal
    %Position(1 --> 0.29° ; 2 --> 0.58°)

    if length (varargin{1})>1

        %vector mode
        cwmargin = varargin{1};
        if isempty(arb.nservos)
            error('RTB:Arbotix:notspec', 'Number of servos not
                specified');
        end
        if length(cwmargin)~=arb.nservos
            error('RTB:Arbotix:badarg', 'Length of CWMARGIN
                vector must match number of servos');
        end
        for j=1:arb.nservos

            arb.writedata1(j,Arbotix.ADDR_CW_COMPLIANCE_MARGIN,
                cwmargin);

        end
    else

        %single joint mode
        id = varargin{1}; cwmargin = varargin{2};

        arb.writedata1(id,Arbotix.ADDR_CW_COMPLIANCE_MARGIN,
            cwmargin);
    end
end

function setccwmargin(arb,varargin)
    %Arbotix.setccwmargin
    %
    %ARB.SETCCWMARGIN(ID,CCWMARGIN) sets the ccw compliance
    %margin [0,255] of servo ID

    %ARB.SETCCWMARGIN(CCWMARGIN) sets the ccw compliance
    %margin of servos 1-N to corresponding elements of the
    %vector POS(1xN).
    %

```

```

%Notes:
%- ID is in the range 1 to N
%- N is defined at construction time by the 'nservos'
%option
%- CCW compliance margin varies from 0 to 255. It means
%the error between goal position and present position in
%the counter clock wise direction. The unit is the same as
%Goal Position (1 --> 0.29° ; 2 --> 0.58°)

if length (varargin{1})>1

    %vector mode
    ccwmargin = varargin{1};
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end
    if length(ccwmargin)~=arb.nservos
        error('RTB:Arbotix:badarg', 'Length of CCWMARGIN
            vector must match number of servos');
    end
    for j=1:arb.nservos
        arb.writedatal(j,Arbotix.ADDR_CCW_COMPLIANCE_MARGIN,
            ccwmargin);
    end
else

    %single joint mode
    id = varargin{1};
    ccwmargin = varargin{2};

    arb.writedatal(id,Arbotix.ADDR_CCW_COMPLIANCE_MARGIN,
        ccwmargin);
end

function setcwslope(arb,varargin)
%Arbotix.setcwslope
%
%ARB.SETCWSLOPE(ID,CWSLOPE) sets the cw compliance slope
%[0,255] of servo ID
%
%ARB.SETCWSLOPE(CWSLOPE) sets the cw compliance slope of
%servos 1-N to corresponding elements of the vector
%POS(1xN).
%
%Notes:
%- ID is in the range 1 to N
%- N is defined at construction time by the 'nservos'
%option
%- CW compliance slope sets the level of torque near the
%goal position. Compliance slope is set in 7 steps, the
%higher value, the more flexibility (smoother movement) is
%obtained. Data representative value is actually used

```

%value. That is, even if the value is set to 25, 16 is
 %used internally as the representative value.

```
%
```

Step	Data value	Data Representative Value
1	0 (0X00) ~3 (0X03)	2 (0x02)
2	4 (0x04) ~ 7 (0x07)	4 (0x04)
3	8 (0x08) ~15 (0x0F)	8 (0x08)
4	16 (0x10) ~31 (0x1F)	16 (0x10)
5	32 (0x20) ~63 (0x3F)	32 (0x20)
6	64 (0x40) ~127 (0x7F)	64 (0x40)
7	128 (0x80) ~254 (0xFE)	128 (0x80)

```
%
```

```
if length (varargin{1})>1

    %vector mode
    cwslope = varargin{1};
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end
    if length(cwslope)~=arb.nservos
        error('RTB:Arbotix:badarg', 'Length of CWSLOPE
            vector must match number of servos');
    end
    for j=1:arb.nservos
        arb.writedata1(j,Arbotix.ADDR_CW_COMPLIANCE_SLOPE,
            cwslope);
    end

else

    %single joint mode
    id = varargin{1};
    cwslope = varargin{2};
    arb.writedata1(id,Arbotix.ADDR_CW_COMPLIANCE_SLOPE,
        cwslope);

end
end
```

```
function setccwslope (arb,varargin)
%Arbotix.setccwslope
%
%ARB.SETCCWSLOPE (ID,CCWSLOPE) sets the ccw compliance
%slope[0,255] of servo ID
%
%ARB.SETCCWSLOPE (CCWSLOPE) sets the ccw compliance slope
%of servos 1-N to corresponding elements of the vector
%POS (1xN) .
%
%Notes:
%- ID is in the range 1 to N
```

```

%- N is defined at construction time by the 'nservos'
%option
%- CCW compliance slope sets the level of torque near the
%goal position. Compliance slope is set in 7 steps, the
%higher value, the more flexibility (smoother movement) is
%obtained. Data representative value is actually used
%value. That is, even if the value is set to 25, 16 is
%used internally as the representative value.
%

```

Step	Data value	Data Representative Value
1	0 (0X00) ~3 (0X03)	2 (0x02)
2	4 (0x04) ~ 7 (0x07)	4 (0x04)
3	8 (0x08) ~15 (0x0F)	8 (0x08)
4	16 (0x10) ~31 (0x1F)	16 (0x10)
5	32 (0x20) ~63 (0x3F)	32 (0x20)
6	64 (0x40) ~127 (0x7F)	64 (0x40)
7	128 (0x80) ~254 (0xFE)	128 (0x80)

```

if length (varargin{1})>1

    %vector mode
    ccwslope = varargin{1};
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end
    if length(ccwslope)~=arb.nservos
        error('RTB:Arbotix:badarg', 'Length of CCWSLOPE
            vector must match number of servos');
    end
    for j=1:arb.nservos

        arb.writedata1(j,Arbotix.ADDR_CCW_COMPLIANCE_SLOPE,
            ccwslope);

    end

else
    %single joint mode
    id = varargin{1};
    ccwslope = varargin{2};

    arb.writedata1(id,Arbotix.ADDR_CCW_COMPLIANCE_SLOPE,
        ccwslope);
end
end

```

```

function setpunch(arb,varargin)
%Arbotix.setpunch
%
%ARB.SETPUNCH(ID,PUNCH) sets the punch [32 (0x20), 1023
%(0x3FF)] of servo ID

```

```

%ARB.SETPUNCH(PUNCH) sets the punch of servos 1-N to
%corresponding elements of the vector POS(1xN).
%
%Notes:
%- ID is in the range 1 to N
%- N is defined at construction time by the 'nservos'
%option
%- Punch varies from [32 to 1023]. It's the minimum
%current to drive motor.
%

if length (varargin{1})>1

    %vector mode
    punch = varargin{1};
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end
    if length(punch)~=arb.nservos
        error('RTB:Arbotix:badarg', 'Length of punch vector
            must match number of servos');
    end
    for j=1:arb.nservos
        arb.writedata2(j, Arbotix.ADDR_PUNCH, punch);
    end

else

    %single joint mode
    id = varargin{1};
    punch = varargin{2};
    arb.writedata2(id, Arbotix.ADDR_PUNCH, punch);
end

end

%% This code belongs to Peter Corke
function setpos(arb, varargin)
%Arbotix.setpos Set position
%
% ARB.SETPOS(ID, POS) sets the position (0-1023) of servo
%ID.
% ARB.SETPOS(ID, POS, SPEED) as above but also sets the
%speed.
%
% ARB.SETPOS(POS) sets the position of servos 1-N to
%corresponding elements of the vector POS (1xN).

% ARB.SETPOS(POS, SPEED) as above but also sets the
%velocity SPEED (1xN).
%
% Notes::
% - ID is in the range 1 to N

```

```

% - N is defined at construction time by the 'nservos'
%option.
% - SPEED varies from 0 to 1023, 0 means largest possible
%speed.
% - POS must be steps of encoder [0-1023].

if length(varargin{1}) > 1

    % vector mode
    pos = varargin{1};

    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end
    if length(pos) ~= arb.nservos
        error('RTB:Arbotix:badarg', 'Length of POS vector
            must match number of servos');
    end

    % First of all, we MUST write SPEED
    % need a separate write for this, since pypose writes
    %max of 2 bytes
    if nargin == 3
        speed = varargin{2};
        if length(speed) ~= arb.nservos
            error('RTB:Arbotix:badarg', 'Length of SPEED
                vector must match number of servos');
        end
        for j=1:arb.nservos
            arb.writedata2(j, Arbotix.ADDR_SPEED, speed(j));
        end
    end

    %NOW WRITE POSE
    for j=1:arb.nservos
        arb.writedata2(j, Arbotix.ADDR_GOAL, pos(j));
    end

else
    % single joint mode
    id = varargin{1}; pos = varargin{2};

    %Write first speed
    if nargin == 4
        speed = varargin{3};
        arb.writedata2(id, Arbotix.ADDR_SPEED, speed);
    end

    % Now write pose
    arb.writedata2(id, Arbotix.ADDR_GOAL, pos);

end
end

```

```

function setpath(arb, jt, t)
    %Arbotix.setpath Load a path into Arbotix controller
    %
    % ARB.setpath(JT) stores the path JT (PxN) in the Arbotix
    %controller where P is the number of points on the path
    %and N is the number of robot joints. Allows for smooth
    %multi-axis motion.
    %
    % See also Arbotix.a2e.

    % will the path fit in Arbotix memory?
    if numrows(jt) > 30
        error('RTB:Arbotix:badarg', 'Too many points on
            trajectory (30 max)');
    end

    jt = Arbotix.a2e(jt);    % convert to encoder values

    % set the number of servos

    arb.command(253, Arbotix.INS_ARB_SIZE_POSE,
        uint8(numcols(jt)));

    pause(0.2);    % this delay is important

    % set the poses
    % payload: <pose#> q1 q2 .. qN
    for i=1:numrows(jt)
        pose = jt(i,:);
        arb.command(253, Arbotix.INS_ARB_LOAD_POSE, [i-1
            typecast( uint16(pose), 'uint8')]);
    end

    % set the sequence in which to execute the poses
    if nargin < 3
        dt = 200;    % milliseconds between poses
    else
        dt = t*1000;
    end

    dt8 = typecast( uint16(dt), 'uint8');
    cmd = [];
    for i=1:numrows(jt)
        cmd = [cmd uint8(i-1) dt8];
    end

    cmd = [cmd 255 0 0];    % mark end of sequence
    arb.command(253, Arbotix.INS_ARB_LOAD_SEQ, cmd);

    % now do it
    arb.command(253, Arbotix.INS_ARB_PLAY_SEQ);

end

```

```

function relax(arb, id, status)
    %Arbotix.relax Control relax state
    %
    % ARB.RELAX(ID) causes the servo ID to enter zero-torque
    %(relax state)
    % ARB.RELAX(ID, FALSE) causes the servo ID to enter
    %position-control mode
    % ARB.RELAX([]) causes servos 1 to N to relax
    % ARB.RELAX() as above
    % ARB.RELAX([], FALSE) causes servos 1 to N to enter
    %position-control mode.
    %
    % Notes::
    % - N is defined at construction time by the 'nservos'
    %option.

    if nargin == 1 || isempty(id)

        % vector mode
        if isempty(arb.nservos)
            error('RTB:Arbotix:notspec', 'Number of servos not
                specified');
        end
        if nargin < 3
            % relax mode
            for j=1:arb.nservos
                arb.writedatal(j, Arbotix.ADDR_TORQUE, 0);
            end
        else
            for j=1:arb.nservos
                arb.writedatal(j, Arbotix.ADDR_TORQUE,
                    status==false);
            end
        end
    end

    else

        % single joint mode
        if nargin < 3
            % relax mode
            arb.writedatal(id, Arbotix.ADDR_TORQUE, 0);
        else
            arb.writedatal(id, Arbotix.ADDR_TORQUE,
                status==false);
        end
    end
end

function settled(arb, id, status)
    %Arbotix.led Set LEDs on servos
    %
    % ARB.led(ID, STATUS) sets the LED on servo ID to on or
    %off according to the STATUS (logical).
    %
    % ARB.led([], STATUS) as above but the LEDs on servos 1 to
    %N are set.
    %

```



```

% Notes::
% - N is defined at construction time by the 'nservos'
%option.

if isempty(id)

    % vector mode
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end

    for j=1:arb.nservos

        arb.writedata(j, Arbotix.ADDR_LED, status);
    end

else

    % single joint mode

    arb.writedata(id, Arbotix.ADDR_LED, status);
end
end

function t = gettemp(arb, id)
%Arbotix.gettemp Get temperature
%
% T = ARB.GETTEMP(ID) is the tempeature (deg C) of servo
%ID.
%
% T = ARB.GETTEMP() is a vector (1xN) of the temperature
%of servos 1 to N.
%
% Notes::
% - N is defined at construction time by the 'nservos'
%option.

arb.flush();

if nargin == 2
    retval = arb.readdata(id, Arbotix.ADDR_TEMP, 1);
    t = retval.params;
elseif nargin < 2
    if isempty(arb.nservos)
        error('RTB:Arbotix:notspec', 'Number of servos not
            specified');
    end
    t = [];
    for j=1:arb.nservos
        retval = arb.readdata(j, Arbotix.ADDR_TEMP, 1);
        t(j) = retval.params;
    end
end
end
end

```

```

function retval = readdata(arb, id, addr, len)
    %Arbotix.readdata Read byte data from servo control table
    %
    % R = ARB.READDATA(ID, ADDR) reads the successive elements
    %of the servo control table for servo ID, starting at
    %address ADDR. The complete return status in the
    %structure R, and the byte data is a vector in the field
    %'params'.
    %
    % Notes::
    % - ID is in the range 0 to N-1, where N is the number of
    %servos in the system.
    % - If 'debug' was enabled in the constructor then the hex
    %values are echoed to the screen as well as being sent to
    %the Arbotix.
    %
    % See also Arbotix.receive, Arbotix.command.

    arb.command(id, Arbotix.INS_READ_DATA, [addr len]);
    retval = arb.receive();

```

end

```

function writedata1(arb, id, addr, data)
    %Arbotix.writedata1 Write byte data to servo control table
    %
    % ARB.WRITEDATA1(ID, ADDR, DATA) writes the successive
    %elements of DATA to the servo control table for servo ID,
    %starting at address ADDR. The values of DATA must be in
    %the range 0 to 255.
    %
    % Notes::
    % - ID is in the range 0 to N-1, where N is the number of
    %servos in the system.
    % - If 'debug' was enabled in the constructor then the hex
    %values are echoed to the screen as well as being sent to
    %the Arbotix.
    %
    % See also Arbotix.writedata2, Arbotix.command.

    % each element of data is converted to a single byte

    arb.command(id, Arbotix.INS_WRITE_DATA, [addr
    uint8(data)]);

```

end

```

function writedata2(arb, id, addr, data)
    %Arbotix.writedata2 Write word data to servo control table
    %
    % ARB.WRITEDATA2(ID, ADDR, DATA) writes the successive
    %elements of DATA to the servo control table for servo ID,
    %starting at address ADDR. The values of DATA must be in
    %the range 0 to 65535.
    %
    % Notes::

```

```

% - ID is in the range 0 to N-1, where N is the number of
%servos in the system.
% - If 'debug' was enabled in the constructor then the hex
%values are echoed to the screen as well as being sent to
%the Arbotix.
%
% See also Arbotix.writedata1, Arbotix.command.

% each element of data is converted to a two byte value
arb.command(id, Arbotix.INS_WRITE_DATA, [addr typecast(
uint16(data), 'uint8')]);
end

```

```

function out = command(arb, id, instruc, data)
%Arbotix.command Execute command on servo
%
% R = ARB.COMMAND(ID, INSTRUC) executes the instruction
%INSTRUC on servo ID.
%
% R = ARB.COMMAND(ID, INSTRUC, DATA) as above but the
%vector DATA forms the payload of the command message, and
%all numeric values in DATA must be in the range 0 to 255.
%
% The optional output argument R is a structure holding
%the return status.
%
% Notes::
% - ID is in the range 0 to N-1, where N is the number of
%servos in the system.
% - Values for INSTRUC are defined as class properties
%INS_*.
% - If 'debug' was enabled in the constructor then the hex
%values are echoed to the screen as well as being sent to
%the Arbotix.
% - If an output argument is requested the serial channel
%is flushed first.
%
% See also Arbotix.receive, Arbotix.flush.

if nargin > 0
    arb.flush();
end

if isempty(id)
    id = 254;    % 0xFE is broadcast
end

if nargin < 4
    data = [];
end
out = [id length(data)+2 instruc data];
cs = bitcmp(uint8(mod(sum(out), 256)));
out = [255 255 uint8(out) cs];
if arb.debug > 0
    fprintf('send:    ');
    fprintf('0x%02x ', out);
end

```

```

        fprintf('\n');
    end
    fwrite(arb.serPort, out);

    if nargout > 0
        out = arb.receive();
    end

end

function out = flush(robot)
    %Arbotix.flush Flush the receive buffer
    %
    % ARB.FLUSH() flushes the serial input buffer, data is
    %discarded.
    %
    % S = ARB.FLUSH() as above but returns a vector of all
    %bytes flushed from the channel.
    %
    % Notes::
    % - Every command sent to the Arbotix elicits a reply.
    % - The method receive() should be called after every
    %command.
    % - Some Arbotix commands also return diagnostic text
    %information.
    %
    % See also Arbotix.receive, Arbotix.parse.

    %Flush Buffer
    N = robot.serPort.BytesAvailable();
    data = [];
    % this returns a maximum of input buffer size
    while (N ~= 0)
        data = [data; fread(robot.serPort, N)];
        pause(0.1); % seem to need this
        N = robot.serPort.BytesAvailable();
    end

    if nargout > 0
        out = data;
    end

end

function s = receive(arb)
    %Arbotix.receive Decode Arbotix return packet
    %
    % R = ARB.RECEIVE() reads and parses the return packet
    %from the Arbotix and returns a structure with the
    %following fields:
    % id          The id of the servo that sent the message
    % error       Error code, 0 means OK
    % params      The returned parameters, can be a vector of
    %             byte values
    %

```

```

% Notes::
% - Every command sent to the Arbotix elicits a reply.
% - The method receive() should be called after every
%command.
% - Some Arbotix commands also return diagnostic text
%information.
% - If 'debug' was enabled in the constructor then the hex
%values are echoed

%
% See also Arbotix.command, Arbotix.flush.
state = 0;
if arb.debug > 0
    fprintf('receive: ');
end
while true
    c = fread(arb.serPort, 1, 'uint8');
    if arb.debug > 0
        fprintf('0x%02x ', c);
    end
    switch state
        case 0 % expecting first 0xff
            if c == 255
                state = 1;
            end
        case 1 % expecting second 0xff
            if c == 255
                state = 2;
            end
        case 2 % expecting id
            s.id = c;
            state = 3;
        case 3 % expecting length
            len = c;
            count = len-2;
            %It's necessary this conditional statement
            (from Josué David Poma 20/07/2019).
            if count ~= 0
                params = [];
                state = 4;
            else
                state = 0;
            end
        case 4 % expecting error code
            s.error = c;
            state = 5;
        case 5 % expecting parameters
            params = [params c];
            count = count - 1;
            if count == 0
                state = 6;
                s.params = params;
            end
        case 6 % expecting checksum
            cs = bitcmp(uint8( mod(s.id + len +
            sum(params), 256)));
    end
end

```

```

        if arb.debug > 0
            fprintf('[0x%02x]\n', cs);
        end
        if cs ~= c
            fprintf('checksum fail: is %d, should be
                    %d\n', c, cs);
        end
        state = 0;

        break;
    end
end

% Low-level Dynamixel bus functions not supported by pypose
% sketch
% Need to create better code for the Arbotix board
%
% function setpos_sync(arb, pos, speed)
%     % pos, speed are vectors
% end
%
% function discover(arb)
%     % find how many servos in the chain
% end
%
% function ping(arb, id)
%     arb.command(id, 1);
%
%     retval = arb.receive();
%     retval
% end
%
% function regwrite(arb, id, addr, data)
%     arb.command(id, 4, [addr data]);
% end
%
% function action(arb)
%     arb.command(id, 5);
% end
%
% function reset(arb, id)
%     arb.command(id, 6);
% end
%
% function syncwrite(arb, addr, matrix)
%     % one column per actuator
%     arb.command(id, hex2dec('83'));
% end
end

methods(Static)
function a = e2a(e)
    %Arbotix.e2a Convert encoder to angle
    %
    % A = ARB.E2A(E) is a vector of joint angles A
    %corresponding to the vector of encoder values E.

```

```

%
% TODO:
% - Scale factor is constant, should be a parameter to
%constructor.
a = (e - 512) / 512 * 150 / 180 * pi;

end

function e = a2e(a)
%Arbotix.a2e Convert angle to encoder
%
% E = ARB.A2E(A) is a vector of encoder values E
%corresponding to the
% vector of joint angles A.
% TODO:
% - Scale factor is constant, should be a parameter to
%constructor.
e = a / pi * 180 / 150 * 512 + 512;
end

function parse(s)
%Arbotix.parse Parse Arbotix return strings
%
% ARB.PARSE(S) parses the string returned from the Arbotix
%controller and prints diagnostic text. The string S
%contains a mixture of Dynamixel style return packets and
%diagnostic text.
%
% Notes::
% - Every command sent to the Arbotix elicits a reply.
% - The method receive() should be called after every
%command.
% - Some Arbotix commands also return diagnostic text
%information.
%
% See also Arbotix.flush, Arbotix.command.

str = [];

while length(s) > 0
    if s(1) == 255 && s(2) == 255
        % we have a packet
        len = s(4);
        pkt = s(1:4+len);
        s = s(4+len+1:end);
    else
        % we have a regular string character
        str = [str s(1)];
        s = s(2:end);
    end
end
fprintf('str: %s\n', char(str));
end

end
end

```

Experimento_tipo1.m

```
%% Experimento 1
%
% Autor: Josué David Poma, julio 2019
% Universidad de Zaragoza
%
% En este primer experimento se graficará la evolución de las
% coordenadas
% articulares, así como las velocidades al llevar el robot desde un
% punto inicial q1= [512 512 512 512 512] a un punto final q2=[600 600
% 600 600 600]. Y realizaremos un posterior análisis de las gráficas
% en la memoria y comentaremos los problemas que surgieron al ejecutar
% este código.
%
% Movemos el robot a la posición inicial de partida
q1 = [512 512 512 512 512];
v_1 = 100*[1 1 1 1 1];

% Como comentamos en la memoria, nosotros introduciremos el número de
% marcas en vez de radianes.
robot1.setpos(q1,v_1);
pause(10);

% Una vez situados en la posición inicial, mediante un bucle
% comenzamos a leer los datos de posición y velocidad, ejecutaremos
% el movimiento de q1 a q2, y mediante otro bucle leeremos la
% evolución de posición y velocidad.

% Inicialización de variables
r= []; cur_pos1=[]; cur_vell=[]; time_1=[];
consigna=[];

% Lectura de posición y velocidad iniciales
r = 1;
cur_pos1(r,:) = robot1.getpos();
cur_vell(r,:) = robot1.getcurspeed();
consigna(r,:) = 600;
time_1(r) = 0;

r = r +1;
tic
for j=1:10
    cur_pos1(r,:) = robot1.getpos();
    cur_vell(r,:) = robot1.getcurspeed();
    consigna(r,:) = 600;
    time_1(r) = toc;
    r = r +1;
end

% Ejecutamos el movimiento de q1 a q2

q2 = [600 600 600 600 600];
```



```

v_1 = 100*[1 1 1 1 1];
robot1.setpos(q2,v_1); %Una vez que se manda esta orden, MatLab
continua
%ejecutando las siguientes líneas de código de este programa
independientemente
%de si el robot ha alcanzado o no la consigna. Lo cual nos permitirá
%leer la posición y velocidad durante el movimiento.

% Leemos la evolución de la posición y velocidad durante el
movimiento.
for j=1:30
    cur_pos1(r,:) = robot1.getpos();
    cur_vell(r,:) = robot1.getcurspeed();
    consigna(r,:) = 600;
    time_1(r) = toc;
    r = r +1;
end

% Graficamos los vectores de posición y velocidad.
pause(2)
close all

figure(1)
subplot(1,2,1);
hold on
plot(time_1, squeeze(cur_pos1(:,1)), 'DisplayName','q1');
plot(time_1, squeeze(cur_pos1(:,2)), 'DisplayName','q2');
plot(time_1, squeeze(cur_pos1(:,3)), 'DisplayName','q3');
plot(time_1, squeeze(cur_pos1(:,4)), 'DisplayName','q4');
plot(time_1, squeeze(cur_pos1(:,5)), 'DisplayName','q5');
plot(time_1, squeeze(consigna(:,1)), 'DisplayName','consigna');
hold off
xlabel('Time (s)')
ylabel('qi (marcas)')
title(['EVOLUCIÓN DE LAS ARTICULACIONES ROBOT1'])
legend
grid

figure(1)
subplot(1,2,2);
hold on
plot(time_1, squeeze(cur_vell(:,1)), 'DisplayName','v1');
plot(time_1, squeeze(cur_vell(:,2)), 'DisplayName','v2');
plot(time_1, squeeze(cur_vell(:,3)), 'DisplayName','v3');
plot(time_1, squeeze(cur_vell(:,4)), 'DisplayName','v4');
plot(time_1, squeeze(cur_vell(:,5)), 'DisplayName','v5');
hold off
xlabel('Time (s)')
ylabel('qdi (marcas)')
title(['EVOLUCIÓN DE LAS VELOCIDADES ROBOT 1'])
legend
grid

```

Obtencion_datos.m

```
%%OBTENCION_DATOS

% Autor: Josué David Poma, julio 2019
% Universidad de Zaragoza
%
% Este código devuelve el valor de varios registros de ambos robots
%entre ellos, los parámetros que modifican la aceleración angular, el
%error de posición admisible, la temperatura, velocidades...

%%Robot 1
Data_r1 = [];
Data_r1(1,:) = robot1.getcwcompmar();
Data_r1(2,:) = robot1.getccwcompmar();
Data_r1(3,:) = robot1.getcwcompslope();
Data_r1(4,:) = robot1.getccwcompslope();
Data_r1(5,:) = robot1.getspeed(); %MOVING OR SETTING SPEED
Data_r1(6,:) = robot1.getpunch();
Data_r1(7,:) = robot1.gettemp();
Data_r1(8,:) = robot1.getpos(); %Posición expresada en encoder
%Posición expresada en grados
    %Motores 1,2 y 5
    Data_r1(9,1) = round((Data_r1(8,1)*300/1023-150),2);
    Data_r1(9,2) = round((Data_r1(8,2)*300/1023-150),2);
    Data_r1(9,5) = round((Data_r1(8,5)*300/1023-150),2);

    %Motores 3 y 4
    Data_r1(9,3) = round((Data_r1(8,3)*300/1023-240),2);
    Data_r1(9,4) = round((Data_r1(8,4)*300/1023-240),2);

%Posición expresada en radiane
Data_r1(10,:)= round((Data_r1(9,:)/180 * pi),2);

Data_r1(11,:) = robot1.getcurspeed;

Data_r2 = [];
Data_r2(1,:) = robot2.getcwcompmar();
Data_r2(2,:) = robot2.getccwcompmar();
Data_r2(3,:) = robot2.getcwcompslope();
Data_r2(4,:) = robot2.getccwcompslope(); %Moving or setting speed
Data_r2(5,:) = robot2.getspeed();
Data_r2(6,:) = robot2.getpunch();
Data_r2(7,:) = robot2.gettemp();
Data_r2(8,:) = robot2.getpos(); %Posición expresada en encoder

    %Posición expresada en grados
    %Motores 1,2 y 5
    Data_r2(9,1) = round((Data_r2(8,1)*300/1023-150),2);
    Data_r2(9,2) = round((Data_r2(8,2)*300/1023-150),2);
    Data_r2(9,5) = round((Data_r2(8,5)*300/1023-150),2);

    %Motores 3 y 4
    Data_r2(9,3) = round((Data_r2(8,3)*300/1023-240),2);
    Data_r2(9,4) = round((Data_r2(8,4)*300/1023-240),2);
```

```

    Data_r2(10,:) = round((Data_r2(9,:)/180 * pi),2); %Posición
expresada en radianes
    Data_r2(11,:) = robot2.getcurspeed;

    Data_r1_str = num2str(Data_r1); %Convertimos a una matriz de
caracteres
    Data_r2_str = num2str(Data_r2); %Convertimos a una matriz de
caracteres

    X = ['DATOS:']; disp(X);
    X = ['R1_CW COMPLIANCE MARGIN : ', Data_r1_str(1,:)];
disp(X);
    X = ['R2_CW COMPLIANCE MARGIN : ', Data_r2_str(1,:)];
disp(X);
    X = ['-----
-----
-']; disp(X);
    X = ['R1_CCW COMPLIANCE MARGIN : ', Data_r1_str(2,:)];
disp(X);
    X = ['R2_CCW COMPLIANCE MARGIN : ', Data_r2_str(2,:)];
disp(X);
    X = ['-----
-----
-']; disp(X);
    X = ['R1_CW COMPLIANCE SLOPE : ', Data_r1_str(3,:)];
disp(X);
    X = ['R2_CW COMPLIANCE SLOPE : ', Data_r2_str(3,:)];
disp(X);
    X = ['-----
-----
-']; disp(X);
    X = ['R1_CCW COMPLIANCE SLOPE : ', Data_r1_str(4,:)];
disp(X);
    X = ['R2_CCW COMPLIANCE SLOPE : ', Data_r2_str(4,:)];
disp(X);
    X = ['-----
-----
-']; disp(X);
    X = ['R1_SETTING SPEED : ', Data_r1_str(5,:)];
disp(X);
    X = ['R2_SETTING SPEED : ', Data_r2_str(5,:)];
disp(X);
    X = ['-----
-----
-']; disp(X);
    X = ['R1_PUNCH : ', Data_r1_str(6,:)];
disp(X);
    X = ['R2_PUNCH : ', Data_r2_str(6,:)];
disp(X);
    X = ['-----
-----
-']; disp(X);
    X = ['R1_TEMPERATURE : ', Data_r1_str(7,:)];
disp(X);
    X = ['R2_TEMPERATURE : ', Data_r2_str(7,:)];
disp(X);

```

```

X = ['-----
-----
-']; disp(X);
X = ['R1_CURRENT POSE (encoder): ', Data_r1_str(8,:)];
disp(X);
X = ['R2_CURRENT POSE (encoder): ', Data_r2_str(8,:)];
disp(X);
X = ['-----
-----
-']; disp(X);
X = ['R1_CURRENT POSE (degree): ', Data_r1_str(9,:)];
disp(X);
X = ['R2_CURRENT POSE (degree): ', Data_r2_str(9,:)];
disp(X);
X = ['-----
-----
-']; disp(X);
X = ['R1_CURRENT POSE (radian): ', Data_r1_str(10,:)];
disp(X);
X = ['R2_CURRENT POSE (radian): ', Data_r2_str(10,:)];
disp(X);
X = ['-----
-----
-']; disp(X);
X = ['R1_CURRENT SPEED : ', Data_r1_str(11,:)];
disp(X);
X = ['R2_CURRENT SPEED : ', Data_r2_str(11,:)];
disp(X);

```

Experimento_tipo2.m

```
%% EXPERIMENTO 2
%
% Autor: Josué David Poma, agosto 2019
% Universidad de Zaragoza
%
% En este segundo experimento se graficará la evolución de las
% coordenadas
% articulares, así como las velocidades al llevar ambos robots desde
% un punto inicial q1= [512 512 512 512 512] a un punto final q2=[600
% 600 600 600]. Pero con la diferencia de configurar un diferente
% compliance slope, el robot1 tendrá un valor de 32 (valor de fábrica)
% mientras que el robot2 tendrá un valor de 64 (menor pendiente).
%

% Configuramos el compliance slope
compliance_slope = 64*[1 1 1 1 1];
compliance_slope_fabrica = 32 * [1 1 1 1 1];
%ROBOT 1
robot1.setcwslope(compliance_slope_fabrica);
robot1.setccwslope(compliance_slope_fabrica);
%ROBOT 2
robot2.setcwslope(compliance_slope);
robot2.setccwslope(compliance_slope);

% Movemos los 2 robots a la posición inicial de partida
q1 = [512 512 512 512 512];
v_1 = 50*[1 1 1 1 1];
% Como comentamos en la memoria, nosotros introduciremos el número de
% marcas en vez de radianes.
robot1.setpos(q1,v_1);
robot2.setpos(q1,v_1);
pause(10);

% Una vez situados en la posición inicial, mediante un bucle
% comenzamos a leer los datos de posición y velocidad, ejecutaremos
% el movimiento de q1 a q2, y mediante otro bucle leeremos la
% evolución de posición y velocidad.

% Inicialización de variables
r= []; cur_pos1=[]; cur_vel1=[]; time=[];
    cur_pos2=[]; cur_vel2=[];
consigna=[];

% Lectura de posición y velocidad iniciales
r = 1;
cur_pos1(r,:) = robot1.getpos();
cur_vel1(r,:) = robot1.getcurspeed();
cur_pos2(r,:) = robot2.getpos();
cur_vel2(r,:) = robot2.getcurspeed();
consigna(r,:) = 600;
time(r) = 0;
```

```

r = r +1;
tic
for j=1:10
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    consigna(r,:) = 600;
    time(r) = toc;
    r = r +1;
end

% Ejecutamos el movimiento de q1 a q2

q2 = [600 600 600 600 600];
robot1.setpos(q2,v_1);
robot2.setpos(q2,v_1);

% Leemos la evolución de la posición y velocidad durante el
movimiento.
for j=1:30
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    consigna(r,:) = 600;
    time(r) = toc;
    r = r +1;
end

% Graficamos los vectores de posición y velocidad.
pause(2)
close all

figure(1)
subplot(2,2,1);
hold on
plot(time, squeeze(cur_pos1(:,1)), 'DisplayName','q1');
plot(time, squeeze(cur_pos1(:,2)), 'DisplayName','q2');
plot(time, squeeze(cur_pos1(:,3)), 'DisplayName','q3');
plot(time, squeeze(cur_pos1(:,4)), 'DisplayName','q4');
plot(time, squeeze(cur_pos1(:,5)), 'DisplayName','q5');
plot(time, squeeze(consigna(:,1)), 'DisplayName','consigna');
hold off
xlabel('Time (s)')
ylabel('qi (marcas)')
title(['EVOLUCIÓN DE LAS ARTICULACIONES ROBOT1'])
legend
grid

figure(1)
subplot(2,2,2);

```

```

hold on
plot(time, squeeze(cur_vel1(:,1)), 'DisplayName', 'v1');
plot(time, squeeze(cur_vel1(:,2)), 'DisplayName', 'v2');
plot(time, squeeze(cur_vel1(:,3)), 'DisplayName', 'v3');
plot(time, squeeze(cur_vel1(:,4)), 'DisplayName', 'v4');
plot(time, squeeze(cur_vel1(:,5)), 'DisplayName', 'v5');
hold off
xlabel('Time (s)')
ylabel('qdi (marcas)')
title(['EVOLUCIÓN DE LAS VELOCIDADES ROBOT 1'])
legend
grid

figure(1)
subplot(2,2,3);
hold on
plot(time, squeeze(cur_pos2(:,1)), 'DisplayName', 'q1');
plot(time, squeeze(cur_pos2(:,2)), 'DisplayName', 'q2');
plot(time, squeeze(cur_pos2(:,3)), 'DisplayName', 'q3');
plot(time, squeeze(cur_pos2(:,4)), 'DisplayName', 'q4');
plot(time, squeeze(cur_pos2(:,5)), 'DisplayName', 'q5');
plot(time, squeeze(consigna(:,1)), 'DisplayName', 'consigna');
hold off
xlabel('Time (s)')
ylabel('qi (marcas)')
title(['EVOLUCIÓN DE LAS ARTICULACIONES ROBOT2'])
legend
grid

figure(1)
subplot(2,2,4);
hold on
plot(time, squeeze(cur_vel2(:,1)), 'DisplayName', 'v1');
plot(time, squeeze(cur_vel2(:,2)), 'DisplayName', 'v2');
plot(time, squeeze(cur_vel2(:,3)), 'DisplayName', 'v3');
plot(time, squeeze(cur_vel2(:,4)), 'DisplayName', 'v4');
plot(time, squeeze(cur_vel2(:,5)), 'DisplayName', 'v5');
hold off
xlabel('Time (s)')
ylabel('qdi (marcas)')
title(['EVOLUCIÓN DE LAS VELOCIDADES ROBOT 2'])
legend
grid

```

Planificacion_recta.m

```
%% GUIADO MANUAL + PLANIFICACIÓN TRAYECTORIA RECTA
%
% Autor: Josué David Poma, septiembre 2019, Universidad de Zaragoza
%
% Mediante guiado manual se obtendrá la primera posición a partir de
%la cual queremos empezar una trayectoria rectilínea.

%Lectura de las posiciones
q1_m = [517 499 744 602 512]; %Aquí escribimos la posición
%que hemos obtenido por guiado manual
q2_m = [517 499 744 602 512];

%Nos quedamos sólo con los cuatro primeros miembros
q1_m = q1_m(:,1:4);
q2_m = q2_m(:,1:4);

%Paso a radianes, aplicamos las ecuaciones del apartado 4.3
for i = 1:4
    switch i
        case 3
            q1(i) = ((pi/3+4/3*pi)/1024)*q1_m(i)-4/3*pi;
            q2(i) = ((pi/3+4/3*pi)/1024)*q2_m(i)-4/3*pi;
        case 4
            q1(i) = ((pi/3+4/3*pi)/1024)*q1_m(i)-4/3*pi;
            q2(i) = ((pi/3+4/3*pi)/1024)*q2_m(i)-4/3*pi;
        otherwise
            q1(i) = (5/6*pi/(1023-512))*q1_m(i)-5/6*pi;
            q2(i) = (5/6*pi/(1023-512))*q2_m(i)-5/6*pi;
    end
end

% Para que las funciones de la cinemática funciones debemos cargar el
%robot, así como la toolbox v9.8. NOTA: La v9.8 se carga
%automáticamente al ejecutar el script Phantom_X.m

restoredefaultpath; %Obligatorio antes de ejecutar PhantomX
Phantom_X;
pause(60); %Obligatorio el pause para que le de tiempo a ejecutarse, ó
usar un breakpoint.

%Cinemática directa, qi-->Xi
T1 = PhantomX.fkine(q1);
T2 = PhantomX.fkine(q2);

round(T1,2);
round(T2,2);

%Y si los graficamos
figure(1)
PhantomX.plot(q1);
```



```

figure(2)
PhantomX.plot(q2);

%subida/bajada
%ROBOT 1: Avanzamos de 145.5 mm a -27.2 mm en el eje z, lo demás
%lo mantenemos igual si dividimos 172 mm en 20 puntos, debemos avanzar
%8.6 mm = 0.0086 metros

%ROBOT2; Avanzamos de 145.5 mm a -8.3 mm en el eje z, lo demás lo
%mantenemos igual, debemos avanzar 154 mm. Si lo dividimos en 20
%puntos nos queda 7.7 mm por cada step= 0.0077metros

%derecha-IZQUIERDA
%robot 2: Avanzamos de 200 mm a 250 mm en el eje x, lo demás lo
%mantenemos igual, si dividimos 50mm en 10 puntos, debemos avanzar
%5 mm= 0.005 m

%robot1: Avanzamos de 200 mm a 150 mm en el eje x, lo demás lo
%mantenemos igual, si dividimos 50 mm en 10 puntos, debemos avanzar -
%5mm = -0.005 m
%

%Subida con la zapatilla deformada
%ROBOT1: Avanzamos de -20 mm a 120 mm en el eje z, lo demás lo
%mantenemos igual, debemos avanzar 120 mm. Si lo dividimos en 10
%puntos nos queda en 12 mm por cada step = 0.012 metros

%ROBOT2: Avanzamos de 0 mm a 120 mm en el eje z, lo demás lo
%mantenemos igual. Debemos avanzar 120 mm. Si lo dividimos en 10
%puntos nos queda en 12 mm por cada step = 0.012 metros.

Points_T1 = []; Points_T2 = [];
Points_T1(:, :, 1)=T1;
Points_T2(:, :, 1)=T2;
for i=2:21
    Points_T1(:, :, i)=Points_T1(:, :, i-1)+[ 0 0 0 0; 0 0 0 0; 0 0 0
-0.0086; 0 0 0 0];
    Points_T2(:, :, i)=Points_T2(:, :, i-1)+[ 0 0 0 0; 0 0 0 0; 0 0 0
-0.0077; 0 0 0 0];
end

traj1_ro1 = [];
traj1_ro2 = [];

%Aplicamos la cinemática inversa
for i=1:21
    traj1_ro1(i, :) = ikinePhantomX(PhantomX,
Points_T1(:, :, i), 'u');
    traj1_ro2(i, :) = ikinePhantomX(PhantomX,
Points_T2(:, :, i), 'u');
end

```

```

%Comprobamos mediante secuencia de dibujos
close all;
figure(1);
for i=1:21% <-- OJO CAMBIAR
    figure(1)
    PhantomX.plot(traj1_ro1(i,:));
    hold on

plot3(Points_T1(1,4,i),Points_T1(2,4,i),Points_T1(3,4,i),'r+:');

end

%Lo expresamos en marcas

for i=1:21
    for j=1:2
        traj1_ro1(i,j)=traj1_ro1(i,j)*195.2+512;
        traj1_ro2(i,j)=traj1_ro2(i,j)*195.2+512;
    end
    for j=3:4
        traj1_ro1(i,j)=traj1_ro1(i,j)*194+814;
        traj1_ro2(i,j)=traj1_ro2(i,j)*194+814;
    end
end

% Añadimos la posición del quinto motor, que en nuestro caso es
512
for i=1:21
    aux1(i,:) = [traj1_ro1(i,:),512];
    aux2(i,:) = [traj1_ro2(i,:),512];
end

traj1_ro1=aux1;
traj1_ro2=aux2;

%redondeamos
traj1_ro1 = round(traj1_ro1)
traj1_ro2 = round(traj1_ro2)

```

Aplicación.m

```
%% ROTACION DE LA PLANTILLA CON ROBOTS COLABORATIVOS
%
% Autor: Josué David Poma, septiembre 2019
% Universidad de Zaragoza
%
%Antes de iniciar el movimiento, se debe ejecutar:
%inicializacion_comunicacion.m

%%MODIFICACION DEL COMPLIANCE SLOPE
%Reducimos a la mitad la pendiente de la rapidez con la que se aplica
%el par
compliance_slope = 64*[1 1 1 1 1];
robot2.setccwslope(compliance_slope);
robot2.setcwslope(compliance_slope);
robot1.setcwslope(compliance_slope);
robot1.setccwslope(compliance_slope);

% Inicialización de variables
    pause(2);
    a= []; b=[]; m = []; n=[]; j=[]; k=[];
    i =[]; r=[]; state = []; e1 = []; e2 = [];

    time_1 = []; time_2 = [];
    cur_pos1 = []; cur_pos2 = [];
    cur_vel1 = []; cur_vel2 = [];

%Hallamos posiciones y velocidades actuales antes
%de que inicie el movimiento
r = 1;
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    time_1(r) = 0;

    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    time_2(r) = 0;

bucle = 1; state = 0; r=2;

% error máximo admisible
N_marcas = 20;
tic
while bucle==1
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    time_1(r) = toc;

    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    time_2(r) = toc;

    r = r + 1;
```

```

switch state
case 0 %Partiendo a la posición de reposo
v_1 = 100*[1 1 1 1 1];
v_2 = 100*[1 1 1 1 1];
pos0 = [512          512          814          814
512];

robot1.setpos(pos0, v_1);
robot2.setpos(pos0, v_2);
i = 0;
while i == 0
cur_pos1(r,:) = robot1.getpos();
cur_vel1(r,:) = robot1.getcurspeed();
time_1(r) = toc;

cur_pos2(r,:) = robot2.getpos();
cur_vel2(r,:) = robot2.getcurspeed();
time_2(r) = toc;

e1 = max(abs(pos0-cur_pos1(r,:)));
e2 = max(abs(pos0-cur_pos2(r,:)));
r = r + 1;

if (e1 < N_marcas) && (e2 < N_marcas)
i = 1;
end
end

pause(30);
state = 1;

case 1 %Partiendo a la posición de arriba
v_1 = 100*[1 1 1 1 1];
v_2 = 100*[1 1 1 1 1];
pos1=[516  498  749  605  512];
robot1.setpos(pos1, v_1);
robot2.setpos(pos1, v_2);
i=0;
while i == 0
cur_pos1(r,:) = robot1.getpos();
cur_vel1(r,:) = robot1.getcurspeed();
time_1(r) = toc;

cur_pos2(r,:) = robot2.getpos();
cur_vel2(r,:) = robot2.getcurspeed();
time_2(r) = toc;

e1 = max(abs(pos1-cur_pos1(r,:)));
e2 = max(abs(pos1-cur_pos2(r,:)));
r = r + 1;

if (e1 < N_marcas) && (e2 < N_marcas)
i = 1;
end
end

```

```
end
end
```

```
state = 2;
```

```
case 2 %Descendiendo en línea recta para posicionarse cerca
%de la plantilla
```

```
v_1 = 100*[ 1 1 1 1 1 ];
```

```
v_2 = 100*[1 1 1 1 1];
```

```
traj_r2 = [ 517 499 744 602 512
517 493 765 587 512
517 489 783 573 512
517 486 800 559 512
517 483 817 545 512
517 482 832 531 512
517 481 846 517 512
517 482 860 503 512
517 484 872 489 512
517 487 884 474 512
517 491 895 459 512
517 496 905 444 512
517 503 914 428 512
517 511 923 412 512
517 520 930 395 512
517 531 936 378 512
517 543 941 361 512
517 557 945 343 512
517 571 948 326 512];
```

```
traj_r1 = [ 517 499 744 602 512
517 493 767 585 512
517 488 787 569 512
517 485 806 554 512
517 482 824 538 512
517 482 840 523 512
517 482 856 507 512
517 483 870 491 512
517 486 883 475 512
517 491 895 458 512
517 497 907 441 512
517 505 917 423 512
517 514 926 405 512
517 526 933 386 512
517 539 940 367 512
517 553 944 348 512
517 569 948 328 512
517 587 949 309 512
517 605 949 291 512
517 625 948 273 512
517 644 945 257 512];
```

```
a= []; b=[]; m = []; n=[]; j=[]; k=[];
```

```
a = size(traj_r1);
```

```

n = a(1);

b = size(traj_r2);
m = b(1);

j = 1; robot1.setpos(traj_r1(j,:) , v_1); n = n - 1;
k = 1; robot2.setpos(traj_r2(k,:) , v_2); m = m - 1;

i = 0; e1 = []; e2= [];

while i==0
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    time_1(r) = toc;

    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    time_2(r) = toc;

    e1 = max(abs(traj_r1(j, :)-cur_pos1(r, :)));
    e2 = max(abs(traj_r2(k, :)-cur_pos2(r, :)));

    if (e1 < N_marcas)
        okr1 = 1;
    else
        okr1 = 0;
    end

    if (e2 < N_marcas)
        okr2 = 1;
    else
        okr2 = 0;
    end

    if (e1 < N_marcas) && (n>0) && (okr2 == 1)
        j = j +1;
        robot1.setpos(traj_r1(j,:), v_1);
        n = n -1;
        okr2 = 0;
    end

    if (e2 < N_marcas) && (m>0) && (okr1 == 1)
        k = k + 1;
        robot2.setpos(traj_r2(k,:), v_2);
        m = m - 1;
        okr1 = 0;
    end

    if ((j == a(1)) && (k == b(1)) && (e1 < N_marcas) &&
(e2< N_marcas) && (okr1 == 1) && (okr2 == 1))
        i = 1;
    end

    r = r + 1;

```

```

end

state = 3;

case 3 %Cerramos pinza
robot2.setpos(5,5,75);
robot1.setpos(5,95,75);
i = 0; e1 = []; e2 = [];
while i == 0
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    time_1(r) = toc;

    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    time_2(r) = toc;

    e1 = abs(95-cur_pos1(r,5));
    e2 = abs(5-cur_pos2(r,5));

    if (e1 < N_marcas) && (e2 < N_marcas)
        i = 1;
    end
    r = r+1;
end

state = 4;

case 4 %Subimos
v_1 = 50*[ 1 1 1 1 1 ];
v_2 = 50*[1 1 1 1 1];
N_marcas = 80;

traj_r2 = [ 517 571 948 326 5
517 557 945 343 5
517 543 941 361 5
517 531 936 378 5
517 520 930 395 5
517 511 923 412 5
517 503 914 428 5
517 496 905 444 5
517 491 895 459 5
517 487 884 474 5
517 484 872 489 5
517 482 860 503 5
517 481 846 517 5
517 482 832 531 5
517 483 817 545 5
517 486 800 559 5
517 489 783 573 5
517 493 765 587 5
517 499 744 602 5];

traj_r1 = [ 517 644 945 257 95
517 625 948 273 95

```

```

517 605 949 291 95
517 587 949 309 95
517 569 948 328 95
517 553 944 348 95
517 539 940 367 95
517 526 933 386 95
517 514 926 405 95
517 505 917 423 95
517 497 907 441 95
517 491 895 458 95
517 486 883 475 95
517 483 870 491 95
517 482 856 507 95
517 482 840 523 95
517 482 824 538 95
517 485 806 554 95
517 488 787 569 95
517 493 767 585 95
517 499 744 602 95];

```

```
a= []; b=[]; m = []; n=[]; j=[]; k=[];
```

```
a = size(traj_r1);
n = a(1);
```

```
b = size(traj_r2);
m = b(1);
```

```
j = 1; robot1.setpos(traj_r1(j,:) , v_1); n = n - 1;
k = 1; robot2.setpos(traj_r2(k,:) , v_2); m = m - 1;
```

```
i = 0; e1 = []; e2= [];
```

```

while i==0
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    time_1(r) = toc;

    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    time_2(r) = toc;

    e1 = max(abs(traj_r1(j,:)-cur_pos1(r,:)));
    e2 = max(abs(traj_r2(k,:)-cur_pos2(r,:)));

    if (e1 < N_marcas)
        okr1 = 1;
    else
        okr1 = 0;
    end

    if (e2 < N_marcas)
        okr2 = 1;
    else

```



```

        okr2 = 0;
    end

    if (e1 < N_marcas) && (n>0) && (okr2 == 1)
        j = j + 1;
        robot1.setpos(traj_r1(j,:), v_1);
        n = n - 1;
        okr2 = 0;
    end

    if (e2 < N_marcas) && (m>0) && (okr1 == 1)
        k = k + 1;
        robot2.setpos(traj_r2(k,:), v_2);
        m = m - 1;
        okr1 = 0;
    end

    if ((j == a(1)) && (k == b(1)) && (e1 < N_marcas) &&
        (e2 < N_marcas) && (okr1 == 1) && (okr2 == 1))
        i = 1;
    end

    r = r + 1;

end

```

```
state = 5;
```

```

case 5 %Movimiento horizontal
    v_1 = 100*[ 1 1 1 1 1 ];
    v_2 = 100*[1 1 1 1 1];
    traj_r2=[]; traj_r1=[];
    N_marcas = 80;

```

```

derecha    traj_r2 =    [    516    501    743    601    5 %centro a la
                516    510    735    600    5
                516    519    726    600    5
                516    529    716    601    5
                516    538    705    602    5
                516    548    693    604    5
                516    559    680    606    5
                516    570    666    609    5
                516    581    651    614    5
                516    594    633    619    5
                516    608    611    626    5 %derecha al
centro
                516    594    633    619    5
                516    581    651    614    5
                516    570    666    609    5
                516    559    680    606    5
                516    548    693    604    5
                516    538    705    602    5
                516    529    716    601    5
                516    519    726    600    5
                516    510    735    600    5

```

```

        516    501    743    601    5 %centro a la
izq
        516    492    751    601    5
        516    483    759    602    5
        516    475    766    604    5
        516    466    772    606    5
        516    458    778    608    5
        516    450    784    611    5
        516    442    789    614    5
        516    434    793    617    5
        516    426    798    621    5
        516    418    801    625    5 %derecha al
centro
        516    426    798    621    5
        516    434    793    617    5
        516    442    789    614    5
        516    450    784    611    5
        516    458    778    608    5
        516    466    772    606    5
        516    475    766    604    5
        516    483    759    602    5
        516    492    751    601    5
        516    501    743    601    5];

traj_r1 = [ 516    501    743    601    95 %centro a la
derecha
        516    492    751    601    95
        516    483    759    602    95
        516    475    766    604    95
        516    466    772    606    95
        516    458    778    608    95
        516    450    784    611    95
        516    442    789    614    95
        516    434    793    617    95
        516    426    798    621    95
        516    418    801    625    95 %derecha al centro
        516    426    798    621    95
        516    434    793    617    95
        516    442    789    614    95
        516    450    784    611    95
        516    458    778    608    95
        516    466    772    606    95
        516    475    766    604    95
        516    483    759    602    95
        516    492    751    601    95
        516    501    743    601    95 %centro a la izq
        516    510    735    600    95
        516    519    726    600    95
        516    529    716    601    95
        516    538    705    602    95
        516    548    693    604    95
        516    559    680    606    95
        516    570    666    609    95
        516    581    651    614    95
        516    594    633    619    95
        516    608    611    626    95 %izquierda al
centro

```

```

516 594 633 619 95
516 581 651 614 95
516 570 666 609 95
516 559 680 606 95
516 548 693 604 95
516 538 705 602 95
516 529 716 601 95
516 519 726 600 95
516 510 735 600 95
516 501 743 601 95];

```

```
a= []; b=[]; m = []; n=[]; j=[]; k=[];
```

```
a = size(traj_r1);
n = a(1);
```

```
b = size(traj_r2);
m = b(1);
```

```
j = 1; robot1.setpos(traj_r1(j,:) , v_1); n = n - 1;
k = 1; robot2.setpos(traj_r2(k,:) , v_2); m = m - 1;
```

```
i = 0; e1 = []; e2= [];
```

```

while i==0
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    time_1(r) = toc;

    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    time_2(r) = toc;

    e1 = max(abs(traj_r1(j,:)-cur_pos1(r,:)));
    e2 = max(abs(traj_r2(k,:)-cur_pos2(r,:)));

    if (e1 < N_marcas)
        okr1 = 1;
    else
        okr1 = 0;
    end

    if (e2 < N_marcas)
        okr2 = 1;
    else
        okr2 = 0;
    end

    if (e1 < N_marcas) && (n>0) && (okr2 == 1)
        j = j +1;
        robot1.setpos(traj_r1(j,:) , v_1);
        n = n -1;
        okr2 = 0;
    end
end

```

```

    if (e2 < N_marcas) && (m>0) && (okr1 == 1)
        k = k + 1;
        robot2.setpos(traj_r2(k,:), v_2);
        m = m - 1;
        okr1 = 0;
    end

    if ((j == a(1)) && (k == b(1)) && (e1 < N_marcas) &&
(e2 < N_marcas) && (okr1 == 1) && (okr2 == 1))
        i = 1;
    end

    r = r + 1;

end

state = 6;

case 6 %Bajamos
v_1 = 50*[ 1 1 1 1 1 ];
v_2 = 50*[1 1 1 1 1];
N_marcas = 80;

traj_r2 = [ 517 499 744 602 5
517 493 765 587 5
517 489 783 573 5
517 486 800 559 5
517 483 817 545 5
517 482 832 531 5
517 481 846 517 5
517 482 860 503 5
517 484 872 489 5
517 487 884 474 5
517 491 895 459 5
517 496 905 444 5
517 503 914 428 5
517 511 923 412 5
517 520 930 395 5
517 531 936 378 5
517 543 941 361 5
517 557 945 343 5
517 571 948 326 5];

traj_r1 = [ 517 499 744 602 95
517 493 767 585 95
517 488 787 569 95
517 485 806 554 95
517 482 824 538 95
517 482 840 523 95
517 482 856 507 95
517 483 870 491 95

```

```

517 486 883 475 95
517 491 895 458 95
517 497 907 441 95
517 505 917 423 95
517 514 926 405 95
517 526 933 386 95
517 539 940 367 95
517 553 944 348 95
517 569 948 328 95
517 587 949 309 95
517 605 949 291 95
517 625 948 273 95
517 644 945 257 95];

```

```
a = []; b=[]; m = []; n=[]; j=[]; k=[];
```

```
a = size(traj_r1);
n = a(1);
```

```
b = size(traj_r2);
m = b(1);
```

```
j = 1; robot1.setpos(traj_r1(j,:) , v_1); n = n - 1;
k = 1; robot2.setpos(traj_r2(k,:) , v_2); m = m - 1;
```

```
i = 0; e1 = []; e2= [];
```

```

while i==0
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    time_1(r) = toc;

    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    time_2(r) = toc;

    e1 = max(abs(traj_r1(j,)-cur_pos1(r,)));
    e2 = max(abs(traj_r2(k,)-cur_pos2(r,)));

    if (e1 < N_marcas)
        okr1 = 1;
    else
        okr1 = 0;
    end

    if (e2 < N_marcas)
        okr2 = 1;
    else
        okr2 = 0;
    end

    if (e1 < N_marcas) && (n>0) && (okr2 == 1)
        j = j +1;
        robot1.setpos(traj_r1(j,:), v_1);

```

```

        n = n - 1;
        okr2 = 0;
    end

    if (e2 < N_marcas) && (m>0) && (okr1 == 1)
        k = k + 1;
        robot2.setpos(traj_r2(k,:), v_2);
        m = m - 1;
        okr1 = 0;
    end

    if ((j == a(1)) && (k == b(1)) && (e1 < N_marcas) &&
(e2 < N_marcas) && (okr1 == 1) && (okr2 == 1))
        i = 1;
    end

    r = r + 1;

end

state = 7;

case 7 %Subimos
v_1 = 50*[ 1 1 1 1 1 ];
v_2 = 50*[1 1 1 1 1];
N_marcas = 80;

traj_r2 = [ 517 571 948 326 5
517 557 945 343 5
517 543 941 361 5
517 531 936 378 5
517 520 930 395 5
517 511 923 412 5
517 503 914 428 5
517 496 905 444 5
517 491 895 459 5
517 487 884 474 5
517 484 872 489 5
517 482 860 503 5
517 481 846 517 5
517 482 832 531 5
517 483 817 545 5
517 486 800 559 5
517 489 783 573 5
517 493 765 587 5
517 499 744 602 5];

traj_r1 = [ 517 644 945 257 95
517 625 948 273 95
517 605 949 291 95
517 587 949 309 95
517 569 948 328 95
517 553 944 348 95
517 539 940 367 95
517 526 933 386 95
517 514 926 405 95

```

```

517 505 917 423 95
517 497 907 441 95
517 491 895 458 95
517 486 883 475 95
517 483 870 491 95
517 482 856 507 95
517 482 840 523 95
517 482 824 538 95
517 485 806 554 95
517 488 787 569 95
517 493 767 585 95
517 499 744 602 95];

```

```
a = []; b=[]; m = []; n=[]; j=[]; k=[];
```

```
a = size(traj_r1);
n = a(1);
```

```
b = size(traj_r2);
m = b(1);
```

```
j = 1; robot1.setpos(traj_r1(j,:) , v_1); n = n - 1;
k = 1; robot2.setpos(traj_r2(k,:) , v_2); m = m - 1;
```

```
i = 0; e1 = []; e2= [];
```

```
while i==0
```

```
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    time_1(r) = toc;
```

```
    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    time_2(r) = toc;
```

```
e1 = max(abs(traj_r1(j, :)-cur_pos1(r, :)));
e2 = max(abs(traj_r2(k, :)-cur_pos2(r, :)));
```

```
if (e1 < N_marcas)
    okr1 = 1;
else
    okr1 = 0;
end
```

```
if (e2 < N_marcas)
    okr2 = 1;
else
    okr2 = 0;
end
```

```
if (e1 < N_marcas) && (n>0) && (okr2 == 1)
    j = j +1;
    robot1.setpos(traj_r1(j,:), v_1);
    n = n -1;
```

```

        okr2 = 0;
    end

    if (e2 < N_marcas) && (m>0) && (okr1 == 1)
        k = k + 1;
        robot2.setpos(traj_r2(k,:), v_2);
        m = m - 1;
        okr1 = 0;
    end

    if ((j == a(1)) && (k == b(1)) && (e1 < N_marcas) &&
(e2 < N_marcas) && (okr1 == 1) && (okr2 == 1))
        i = 1;
    end

    r = r + 1;

end

state = 8;

case 8 %Vamos a la posición inicial donde la suela está
deformada
    v_1 = 50*[1 1 1 1 1];
    v_2 = 50*[1 1 1 1 1];
    pos1 = [567 581 748 501 95];
    pos2 = [573 612 687 517 5];
    robot1.setpos(pos1, v_1);
    robot2.setpos(pos2, v_2);
    N_marcas = 20;
    i = 0;
    while i == 0
        cur_pos1(r,:) = robot1.getpos();
        cur_vel1(r,:) = robot1.getcurspeed();
        time_1(r) = toc;

        cur_pos2(r,:) = robot2.getpos();
        cur_vel2(r,:) = robot2.getcurspeed();
        time_2(r) = toc;

        e1 = max(abs(pos1-cur_pos1(r,:)));
        e2 = max(abs(pos2-cur_pos2(r,:)));
        r = r + 1;

        if (e1 < N_marcas) && (e2 < N_marcas)
            i = 1;
        end
    end

state=9;

case 9 %Bajamos
    v_1 = 50*[ 1 1 1 1 1 ];

```



```
v_2 = 50*[1 1 1 1 1];
N_marcas = 20;
```

```
traj_r2 = [ 573    612    687    517    5
            573    610    712    494    5
            573    611    733    472    5
            573    614    750    452    5
            573    620    764    433    5
            573    627    775    414    5
            573    637    784    396    5
            573    648    790    378    5
            573    661    794    362    5
            573    675    795    347    5
            573    690    794    333    5];
```

```
traj_r1 = [ 567    581    748    501    95
            567    582    768    480    95
            567    586    784    459    95
            567    592    799    439    95
            567    601    810    419    95
            567    611    819    400    95
            567    623    826    381    95
            567    637    830    363    95
            567    652    832    346    95
            567    668    832    330    95
            567    686    829    316    95];
```

```
a = []; b=[]; m = []; n=[]; j=[]; k=[];
```

```
a = size(traj_r1);
n = a(1);
```

```
b = size(traj_r2);
m = b(1);
```

```
j = 1; robot1.setpos(traj_r1(j,:) , v_1); n = n - 1;
k = 1; robot2.setpos(traj_r2(k,:) , v_2); m = m - 1;
```

```
i = 0; e1 = []; e2= [];
```

```
while i==0
    cur_pos1(r,:) = robot1.getpos();
    cur_vel1(r,:) = robot1.getcurspeed();
    time_1(r) = toc;

    cur_pos2(r,:) = robot2.getpos();
    cur_vel2(r,:) = robot2.getcurspeed();
    time_2(r) = toc;
```

```

e1 = max(abs(traj_r1(j,:)-cur_pos1(r,:)));
e2 = max(abs(traj_r2(k,:)-cur_pos2(r,:)));

if (e1 < N_marcas)
    okr1 = 1;
else
    okr1 = 0;
end

if (e2 < N_marcas)
    okr2 = 1;
else
    okr2 = 0;
end

if (e1 < N_marcas) && (n>0) && (okr2 == 1)
    j = j + 1;
    robot1.setpos(traj_r1(j,:), v_1);
    n = n - 1;
    okr2 = 0;
end

if (e2 < N_marcas) && (m>0) && (okr1 == 1)
    k = k + 1;
    robot2.setpos(traj_r2(k,:), v_2);
    m = m - 1;
    okr1 = 0;
end

if ((j == a(1)) && (k == b(1)) && (e1 < N_marcas) &&
(e2 < N_marcas) && (okr1 == 1) && (okr2 == 1))
    i = 1;
end

r = r + 1;

end
state = 10;

case 10 %abrimos pinzas
    robot2.setpos(5,300,50);
    robot1.setpos(5,300,50);
    i = 0; e1 = []; e2 = [];
    N_marcas = 20;
    while i == 0
        cur_pos1(r,:) = robot1.getpos();
        cur_vel1(r,:) = robot1.getcurspeed();
        time_1(r) = toc;

        cur_pos2(r,:) = robot2.getpos();
        cur_vel2(r,:) = robot2.getcurspeed();
        time_2(r) = toc;
    end
end

```

```

    e1 = abs(300-cur_pos1(r,5));
    e2 = abs(300-cur_pos2(r,5));

    if (e1 < N_marcas) && (e2 < N_marcas)
        i = 1;
    end
    r = r+1;
end

state = 11;

```

```

case 11 %Nos alejamos de la suela
    pos1 = [574 656 953 230 398];
    pos2 = [591 656 877 312 398];
    v_1 = 50*[1 1 1 1 1]; v_2 = v_1;
    robot2.setpos(pos2, v_1);
    robot1.setpos(pos1, v_2);
    i = 0; e1 = []; e2 = [];
    N_marcas = 20;
    while i == 0
        cur_pos1(r,:) = robot1.getpos();
        cur_vel1(r,:) = robot1.getcurspeed();
        time_1(r) = toc;

        cur_pos2(r,:) = robot2.getpos();
        cur_vel2(r,:) = robot2.getcurspeed();
        time_2(r) = toc;

        e1 = max(abs(pos1-cur_pos1(r,:)));
        e2 = max(abs(pos2-cur_pos2(r,:)));

        if (e1 < N_marcas) && (e2 < N_marcas)
            i = 1;
        end
        r = r+1;
    end

    state = 12;

```

```

case 12 %Volvemos a la posición de reposo
    v_1 = 50*[1 1 1 1 1];
    v_2 = 50*[1 1 1 1 1];
    pos0 = [512          512          814          814
512];
    robot1.setpos(pos0, v_1);
    robot2.setpos(pos0, v_2);
    i = 0;
    while i == 0
        cur_pos1(r,:) = robot1.getpos();
        cur_vel1(r,:) = robot1.getcurspeed();

```

```

        time_1(r) = toc;

        cur_pos2(r,:) = robot2.getpos();
        cur_vel2(r,:) = robot2.getcurspeed();
        time_2(r) = toc;

        e1 = max(abs(pos0-cur_pos1(r,:)));
        e2 = max(abs(pos0-cur_pos2(r,:)));
        r = r + 1;

        if (e1 < N_marcas) && (e2 < N_marcas)
            i = 1;
        end
    end

    bucle = 0;

end

end
pause(2)
close all

figure(1)
subplot(2,2,1);
hold on
plot(time_1, squeeze(cur_pos1(:,1)), 'DisplayName','q1');
plot(time_1, squeeze(cur_pos1(:,2)), 'DisplayName','q2');
plot(time_1, squeeze(cur_pos1(:,3)), 'DisplayName','q3');
plot(time_1, squeeze(cur_pos1(:,4)), 'DisplayName','q4');
plot(time_1, squeeze(cur_pos1(:,5)), 'DisplayName','q5');
hold off
xlabel('Time (s)')
ylabel('qi (marcas)')
title(['EVOLUCIÓN DE LAS ARTICULACIONES ROBOT1'])
legend
grid

figure(1)
subplot(2,2,2);
hold on
plot(time_2, squeeze(cur_pos2(:,1)), 'DisplayName','q1');
plot(time_2, squeeze(cur_pos2(:,2)), 'DisplayName','q2');
plot(time_2, squeeze(cur_pos2(:,3)), 'DisplayName','q3');
plot(time_2, squeeze(cur_pos2(:,4)), 'DisplayName','q4');
plot(time_2, squeeze(cur_pos2(:,5)), 'DisplayName','q5');
hold off
xlabel('Time (s)')
ylabel('qi (marcas)')
title(['EVOLUCIÓN DE LAS ARTICULACIONES ROBOT 2'])
legend
grid

figure(1)

```

```

subplot(2,2,3);
hold on
plot(time_1, squeeze(cur_vel1(:,1)), 'DisplayName', 'v1');
plot(time_1, squeeze(cur_vel1(:,2)), 'DisplayName', 'v2');
plot(time_1, squeeze(cur_vel1(:,3)), 'DisplayName', 'v3');
plot(time_1, squeeze(cur_vel1(:,4)), 'DisplayName', 'v4');
plot(time_1, squeeze(cur_vel1(:,5)), 'DisplayName', 'v5');
hold off
xlabel('Time (s)')
ylabel('qdi (marcas)')
title(['EVOLUCIÓN DE LAS VELOCIDADES ROBOT 1'])
legend
grid

```

```

figure(1)
subplot(2,2,4);
hold on
plot(time_2, squeeze(cur_vel2(:,1)), 'DisplayName', 'v1');
plot(time_2, squeeze(cur_vel2(:,2)), 'DisplayName', 'v2');
plot(time_2, squeeze(cur_vel2(:,3)), 'DisplayName', 'v3');
plot(time_2, squeeze(cur_vel2(:,4)), 'DisplayName', 'v4');
plot(time_2, squeeze(cur_vel2(:,5)), 'DisplayName', 'v5');
hold off
xlabel('Time (s)')
ylabel('qdi (marcas)')
title(['EVOLUCIÓN DE LAS VELOCIDADES ROBOT 2'])
legend
grid

```

A.4 Tabla de especificaciones del Dynamixel AX12-A

La tabla que se presenta a continuación ha sido extraída de la página e-manual ROBOTIS [6].

Item	Specifications
Baud Rate	7843 bps ~ 1 Mbps
Resolution	0.29 [°]
Running Degree	0 [°] ~ 300 [°] Endless Turn
Weight	53.5g(AX-12, AX-12+), 54.6g(AX-12A)
Dimensions (W x H x D)	32mm x 50mm x 40mm
Gear Ratio	254 : 1
Stall Torque	1.5 N*m (at 12V, 1.5A)
No Load Speed	59rpm (at 12V)
Operating Temperature	-5 [°C] ~ +70 [°C]
Input Voltage	9.0 ~ 12.0V (Recommended : 11.1V)
Command Signal	Digital Packet
Protocol Type	Half Duplex Asynchronous Serial Communication (8bit, 1stop, No Parity)
Physical Connection	TTL Level Multi Drop Bus
ID	0 ~ 253
Feedback	Position, Temperature, Load, Input Voltage, etc
Material	Engineering Plastic

A.5 Circuito del motor Dynamixel AX12+

Este circuito [22] corresponde a la versión anterior del modelo AX12A, pero el esquema básico es el mismo, por ello, lo mostramos aquí.

