2019

DESARROLLO DE UNA PLATAFORMA ROBÓTICA PARA REALIZAR EXPERIMENTOS DE CONTROL, EN EL PROGRAMA DE INGENIERÍA MECATRÓNICA, UNIVERSIDAD NACIONAL SEDE BOGOTÁ

PASANTÍA DE INVESTIGACIÓN, PREGRADO

JULIAN TOSCANO

DESARROLLO DE UNA PLATAFORMA ROBÓTICA PARA REALIZAR EXPERIMENTOS DE CONTROL, EN EL PROGRAMA DE INGENIERÍA MECATRÓNICA, UNIVERSIDAD NACIONAL SEDE BOGOTÁ

Pasantía de Investigación, Pregrado

autor
Julián Alexander Toscano Pinzón

Director
Ph.D. César Augusto Peña
Universidad de Pamplona

Codirector
Ph.D. Pedro Fabián Cárdenas

INGENIERÍA MECATRÓNICA
DEPARTAMENTO MMI
FACULTAD DE INGENIERÍAS Y ARQUITECTURA

UNIVERSIDAD DE PAMPLONA
2019

RESUMEN

Este trabajo se centra en el desarrollo de una plataforma robótica de 3 GDL para la implementación de esquemas de control. Para llevar a cabo este proyecto se utilizan herramientas para la identificación de la planta, como Simscape que proporciona Mathworks®. Este toolbox para simulink de Matlab, cuenta Simscape Power Systems, Simscape Electronics, Simscape Fluids, Simscape Multibody, Simscape Driveline, Lo cual permite modelar tanto en el entorno físico, eléctrico y electrónico y así obtener la dinámica del sistema de una forma más eficiente y rápida, el diseño del robot fue realizado en el CAD solidworks y mediante el plugin Simscape Multibody se exporta el robot a simulink, con Simscape Electronics proporciona un conjunto de bloques que permite modelar los actuadores, para su respectivo control se recurre a un controlador PID mediante feedback el cual permite valga la redundancia controlar la posición asignada, y para control de velocidad se recurre al control feed-forward, dichos parámetros son proporcionados por el control cinemático el cual establece las trayectorias que debe seguir cada articulación del robot a lo largo del tiempo. Las trayectorias que se implementan en este proyecto son las siguientes, en el caso de trayectorias punto a punto; movimiento eje a eje, movimiento simultáneo de ejes y trayectoria coordinadas o isócronas, en este tipo de trayectorias no importa el camino del extremo del robot, solo importa que alcance el punto final indicado. Para el caso de trayectorias continuas importa el camino, puesto que se pretende que el extremo del robot describa una trayectoria concreta y conocida para poder realizar tareas como corte, soldadura etc. Utilizando la transformación inversa convierte cada punto en su correspondiente valor articular para cada actuador. En este proyecto se implementa splines cúbicos debido a que se dispone de condiciones de posición y velocidad, y este interpolador proporciona un excelente ajuste a los puntos ingresados y sus calculo no son tan complejos. Las actividades extras en las que se participan son las siguientes; asistencia a los cursos de Robótica y laboratorio de robótica, servomecanismos, visión de máquina y automatización de procesos de manufactura, con el fin de proporcional conocimiento para el correcto desarrollo de este proyecto. Cursos dictados por la universidad para estudiantes pregrado como posgrado.

***Palabras claves**: Simscape, paralelogramo, PID, feedback, FeedForward, trayectorias*

ABSTRACT

This work focuses on the development of a 3 GDL robotic platform for the implementation of control schemes. To carry out this project, tools for plant identification are used, such as Simscape provided by Mathworks®. This toolbox for Matlab simulink, features Simscape Power Systems, Simscape Electronics, Simscape Fluids, Simscape Multibody, Simscape Driveline, which allows modeling both in the physical, electrical and electronic environment and thus obtain the dynamics of the system in a more efficient and fast, the robot design was done in the solidworks CAD and through the Simscape Multibody plugin the robot is exported to simulink, with Simscape Electronics it provides a set of blocks that allow the actuators to be modeled, for their respective control a PID controller is used through feedback which allows redundancy to control the assigned position, and for speed control the feed-forward control is used, these parameters are provided by the kinematic control which establishes the trajectories that each robot joint must follow over time . The trajectories that are implemented in this project are the following, in the case of point-to-point trajectories; axis-to-axis movement, simultaneous movement of coordinated or isochronous axes and trajectory, in this type of trajectories the path of the robot's end does not matter, it only matters that it reaches the indicated end point. In the case of continuous trajectories, the path matters, since it is intended that the end of the robot describe a concrete and known trajectory to perform tasks such as cutting, welding, etc. Using the inverse transformation converts each point into its corresponding joint value for each actuator. This project implements cubic splines because of position and speed conditions are available, and this interpolator provides an excellent fit to the points entered and their calculation is not so complex. The extra activities in which they participate are the following; Robotics and robotics laboratory, servomechanisms, machine vision and automation of manufacturing processes courses, in order to provide proportional knowledge for the correct development of this project. Courses taught by the university for undergraduate and graduate students.

Keywords*: Simscape, parallelogram, PID, feedback, FeedForward, trajectories*

Objetivo General

- Desarrollar una plataforma robótica de tres grados de libertad, dirigido a docentes y estudiantes del programa de ingeniería mecatrónica de la universidad nacional sede Bogotá, para ejecutar algoritmos de control.

Objetivos específicos

- Diseñar y construir la estructura robótica de 3 grados de liberta.
- Implementar algoritmos de bajo nivel a los motores de la estructura robótica.
- Desarrollar algoritmos de control cinemático, y control dinámico
- Elaborar las guías de implementación de algoritmos de control de docentes y estudiantes.
- Redacción de artículo científico del proyecto.

CONTENIDO

TABLA DE ILUSTRACIONES

LISTA DE ANEXOS

CAPÍTULO 1, MARCO TEÓRICO Y ESTADO DEL ARTE

1.1 INTRODUCCIÓN

La interacción entre humanos y robots se ha estudiado intensivamente y se han publicado muchos artículos relacionados desde las últimas décadas, en los que se propuso una taxonomía y su versión actualizada de la interacción entre humanos y robots (Holly A & Jill L, 2002) (Yanco & Drury, 2005). La inclusión de robots en entornos se define como la métrica para evaluar en qué medida el diseño del entorno tiene en cuenta el robot que se encuentra en él (Ning, Mohan, & Akiko, 2016). Estos son aspectos que se debe tener en consideración a la hora de diseñar o elegir el mejor tipo de robot, como conceptos que se mencionan más adelante.

El desarrollo de robots colaborativos que tienen que trabajar de manera conjunta con los humanos, se ha convertido en una tendencia tanto en el mundo académico como en la industria, lo cual ha tomado un papel de gran importancia en el desarrollo tecnológico, ya que nos ha permitido llevar tareas que requieren mayor precisión, velocidad y eficiencia, como también reemplazar al hombre en tareas peligrosas evitando que los seres humanos se expongan (Merckaert, y otros, 2018). En el mundo académico como es el caso de este proyecto se centra en enseñar a aplicar diversas herramientas matemáticas, y herramientas que ofrecen los softwares como el caso de Matlab® y Solidworks® que se usan en este proyecto para modelar y expresar la dinámica del sistema, y así integrar la información obtenida, en la selección de los componentes requeridos para la aplicación y llevar acabo el diseño del mismo, como también entender el comportamiento dinámico de un sistema existente, y así poder aplicar diferentes esquemas de control, mostrando de forma abstracta y fácil.

1.2 PLATAFORMA DE HARDWARE LIBRE:

Se les denomina hardware libre a los dispositivos cuyas especificaciones y diagramas esquemáticos son de acceso público, ya sea de algún tipo de pago o de forma gratuita, uno de los más populares es Arduino® (Ghariblu, 2015). Esta plataforma bastante asequible y cuenta con la ventaja de tener una gran comunidad, brindando bastante información de implementación de proyectos, errores y soluciones existentes, además de tener soporte y compatibilidad en

múltiples plataformas como en el caso de Matlab®, el cual es la principal razón de utilizar dicha plataforma.

La plataforma cuenta con diversas placas. la utilizada en este proyecto es la Arduino Mega como se muestra en la figura 1, dispone de una CPU principal: ATmega2560. con 256 KB de memoria flash de los cuales 8 KB utilizados por el gestor de arranque, y 4 KB EEPROM, velocidad de reloj a una frecuencia de 16 MHz, Pines digitales de Entrada/Salida: 54 (de los cuales 15 proveen salida PWM), además cuenta con tres timers razón por el cual se selecciona dicha versión, ya que se utilizan tres encoders en este proyecto (Arduino®, s.f.).

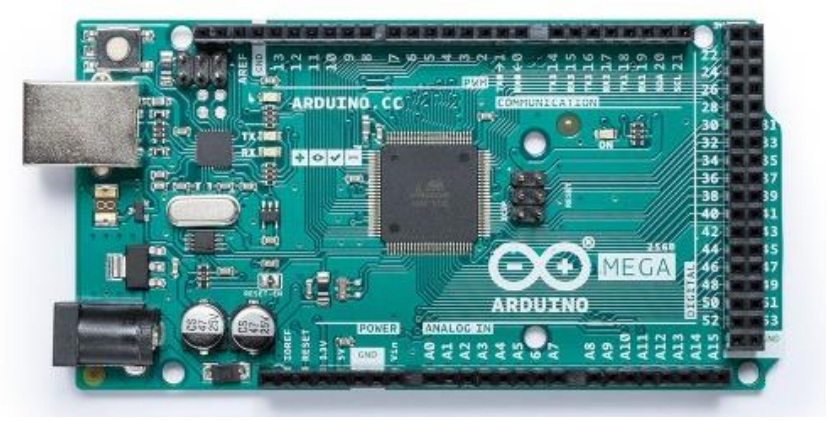


Figure 1. Arduino Mega 2560 REV3 (Arduino®, s.f.).

1.3 ACTUADORES EN ROBÓTICA

Para que exista realimentación de fuerzas se debe disponer de actuadores. Sin ellos tan solo podrían ser dispositivos de entrada de datos. La tecnología actual de los actuadores está ligada al desarrollo de un mejor rendimiento y funcionalidad. Podemos identificar los actuadores clásicos en investigación: Motores eléctricos, accionamientos hidráulicos, accionamientos neumáticos, aleaciones con memoria de forma, actuadores piezoeléctricos, fluidos electro y magneto-reológicos, polímeros electro activos (RUÍZ OLAYA, 2008). Generalmente estos actuadores los encontramos con su respectivo controlador, para el caso de este proyecto se cuenta con motores y controladores de la marca alemana Faulhaber®, ya se contaba con dichos motores y controladores antes del inicio del proyecto por lo cual no había posibilidad de selección. A continuación, se describe los componentes y funcionamiento de los actuadores.

1.3.1 DC-Servomotors series 2036 ... B, con accesorios

Este servo motor cuenta con tecnología de 2 polos, con una longitud de 20 mm y 36 mm, con un torque 7.2 mNm y una potencia de 25 W, la reductora de precisión es de tipo planetario cuenta con una reducción 159/1, el encoder cuenta 64 pulsos por revolución. La figura 2 muestra por separado el conjunto; encoder, caja reductora y motor sin escobillas (Faulhaber®, faulhaber.com, s.f.).

Figure 2. Servomotores de CC sin escobillas serie 2036 ... B , reductores de precisión Serie 20 / 1R, y encoder serie IE2-64 (Faulhaber®, faulhaber.com, s.f.).

1.3.2 Brushless DC-Servomotors serie 1628 ... B, con accesorios

Al igual que al modelo anterior este servo motor cuenta con tecnología de 2 polos, con una longitud de 16 mm y 28 mm, con un torque 3.3 mNm y una potencia de 17 W, la reductora de precisión es de tipo planetario cuenta con una reducción 159/1, el encoder cuenta 64 pulsos por revolución (Faulhaber®, faulhaber.com, s.f.). En la figura 3 se visualiza las partes compone este actuador.

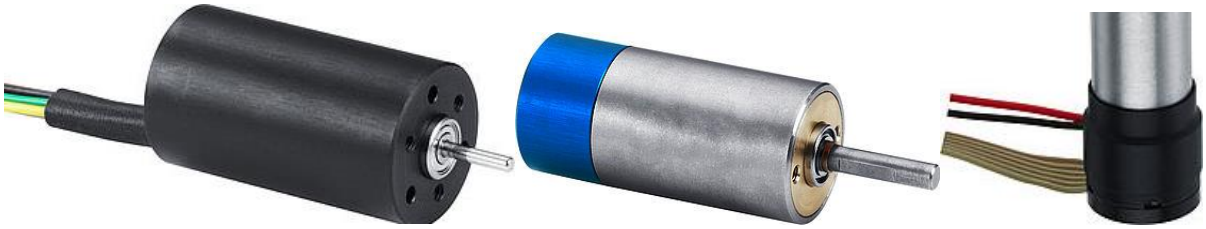


Figure 3. Servomotores de CC sin escobillas serie 1628, reductores de precisión Serie 16 / 7, y encoder serie IE2-64 (Faulhaber®, faulhaber.com, s.f.).

1.3.4 Motion Controller MCBL 2805

Este controlador nos permite comunicarnos mediante puerto CAN y serial, fue diseñado para motores sin escobillas con sensores Hall lineales. El controlador de movimiento se basa en un microcontrolador de 16 bits con una buena calidad de filtración, este controlador de movimiento inteligente realiza las siguientes tareas: Control de Velocidad con perfiles de velocidad; En rampa, triángulo, trapezoidal y perfiles de velocidad combinadas más complicadas están disponibles para el usuario. Modo de posicionamiento; Posicionamiento con alta resolución incluidas finales de carrera y cero de referencia. Modos adicionales; Por ejemplo, modo paso a paso motor para sincronizar múltiples motores. Torque Control; Logrado a través de la regulación actual. En la figura 4, se muestra el esquemático del controlador, al lado derecho se dispone de los pines de salida para los motores indicándonos la alimentación, fases y los pines para los sensores hall clasificados en diferentes colores para mayor facilidad de conexión entre el controlador y el motor. Al lado izquierdo están las entradas del controlador el cual dispone de entra UART Serial (RX, TX), un pin analógico con su respectivo pin de tierra para el control de velocidad o posición como opción, un pin indicación de fallos programable y un pin adicional para comandos, y el pin de alimentación a 24 V y pin de tierra (Faulhaber®, Manual Motion Controller for Brushless DC-Servomotors).

Figure 4. Control de movimiento para servomotores brushless CC ***(Faulhaber®, Manual Motion Controller for Brushless DC-Servomotors).***

La figura 5, muestra un esquema general de los componentes que comprender el controlador, y sus respectivas conexiones con los pines de salida y entrada.

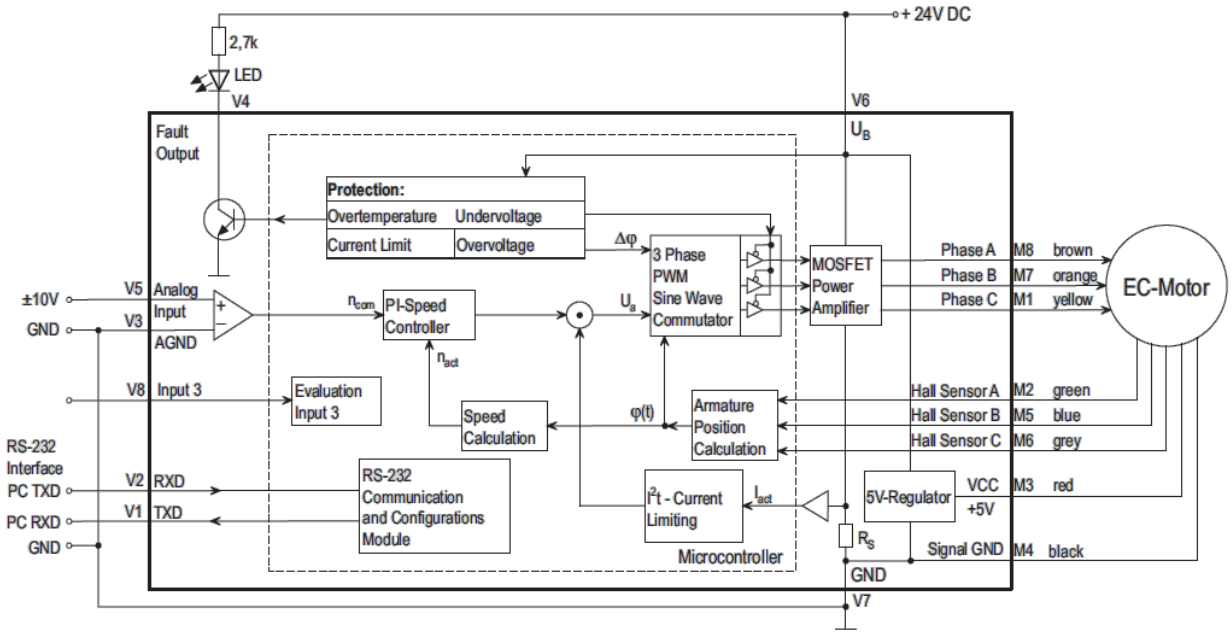


Figure 5. Control de movimiento para servomotores brushless CC ***(Faulhaber®, Manual Motion Controller for Brushless DC-Servomotors).***

Adicionalmente dispone de un puerto de RS-232 que se puede usar para conectar a un pc mediante un convertidor de protocolo de RS-232 a USB, como también para conectarse a otros controladores mediante una tarjeta multiplexor como puente. La programación se realiza mediante un conjunto de comandos ASCII que puede ser enviada desde un terminal desde la PC, igualmente se dispone del software FAULHABER Motion Manager que ofrece el fabricante para la programación y configuración, el controlador dispone de diferentes modos de operación como se indica en el siguiente inciso (Faulhaber®, Manual Motion Controller for Brushless DC-Servomotors).

1.3.4 FAULHABER Motion Manager

Este software proporcionado por el fabricante brinda un cómodo acceso a la configuración y los parámetros de los controles de motores conectados, la interfaz gráfica de usuario como muestra en la figura 6, se utiliza para leer, modificar y volver a cargar las configuraciones. Los comandos individuales o conjuntos, parámetros completos y secuencias de programa se pueden introducir y se transfieren al control. Las respuestas de los comandos se registran junto con los comandos enviados en la pestaña “Historial”. Además, las opciones de análisis están disponibles

en forma de indicadores de estado y las ventanas gráficas (Faulhaber®, Instruction Manual FAULHABER Motion Manager 5.4, 2015).

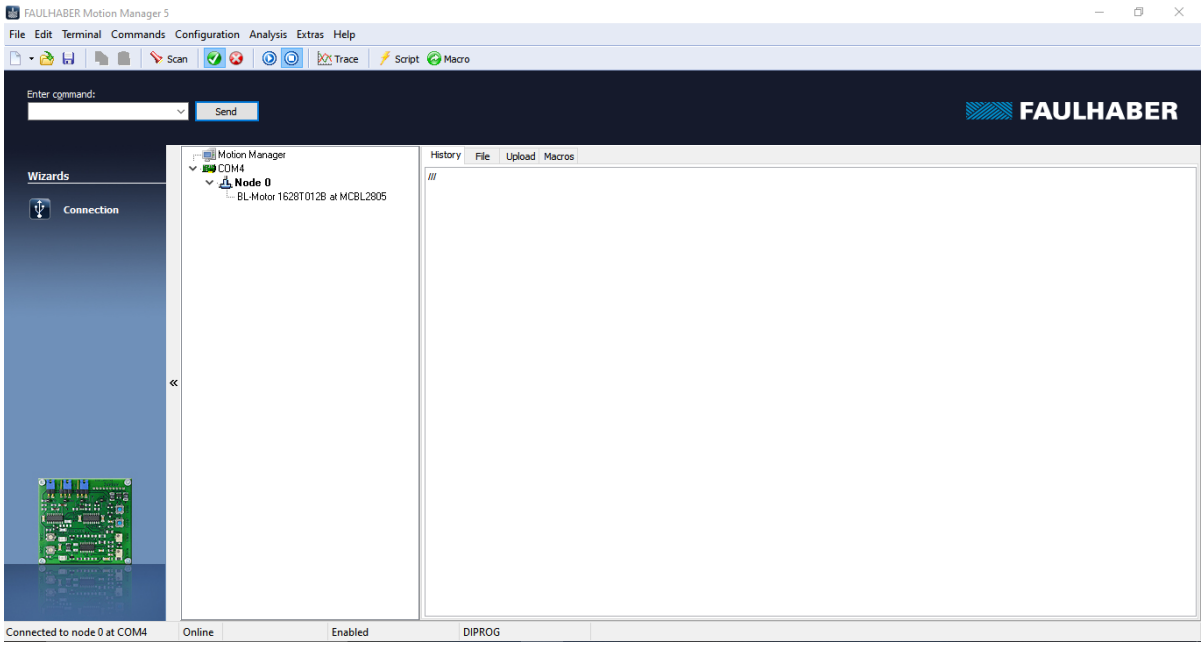


Figure 6. Interfaz gráfica de FAULHABER Motion Manager ***(Faulhaber®, Instruction Manual FAULHABER Motion Manager 5.4, 2015).***

En la figura 7, se muestra el menú de configuración del controlador, las opciones disponibles dependen del controlar conectado para el caso del controlador MCBL 2805 son las siguientes.

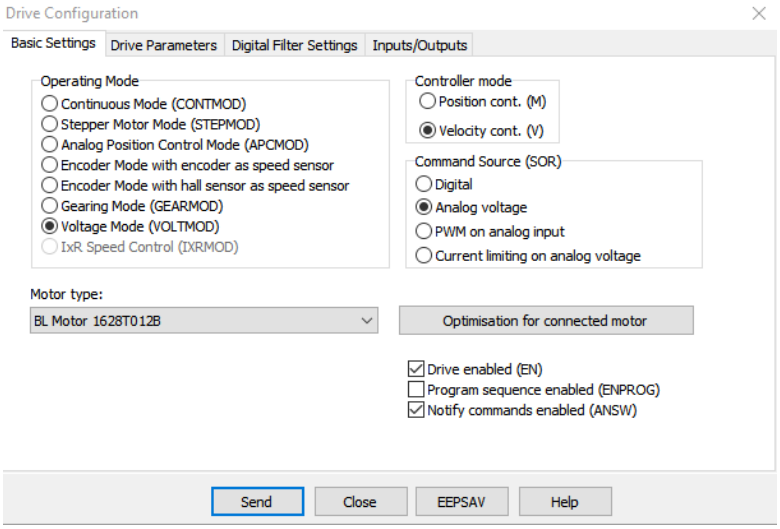


Figure 7. Menú de configuración del controlador ***(Faulhaber®, Instruction Manual FAULHABER Motion Manager 5.4, 2015).***

En las configuraciones básicas tenesmos los modos de operación (en este proyecto se utiliza el modo voltaje), modo de control y la fuente de comandos, optimización para el motor conectado, habilitar secuencias y notificaciones. En parámetros del controlador nos permite configurar el rango giro limite, la desviación permitida, dirección de giro, valores de voltaje mínimo y de inicio en modo análogo de control de velocidad, pasos por revolución en modo motor paso a paso, y la configuración de pulso por revolución del encoder externo. En configuraciones del filtro digital nos permite editar los valores del control PI, el pico de corriente, corriente pico de operación, la aceleración y el máximo de velocidad. En entradas y salidas lo que nos permite es configurar el fault pin en modo de salida de error, salidas de pulsos, salida digital, entrada digital y dirección de rotación. También nos permite configurar el pin análogo, fault y 3 como entrada de control de velocidad o posición, como pin para activar el bloqueo, bloqueo en giro horario, como también activar el cero de referencia, parar el motor si se presenta cambios de giro más allá del límite configurado (Faulhaber®, Instruction Manual FAULHABER Motion Manager 5.4, 2015).

1.4 MODELAR Y SIMULAR SISTEMAS FÍSICOS MULTIDOMINIO. (SIMSCAPE ™)

Simscape™ es una herramienta desarrollada por Mathworks® que permite crear de forma rápida modelos de sistemas físicos dentro del ambiente de Simulink®. Consta de los paquetes de Simscape Electrical; Esta nos permite modelar sistemas eléctricos. Simscape Driveline; sirve para modelar sistemas mecánicos, como transmisiones. Simscape Multibody; el cual nos permite modelar eslabones y juntas, por ejemplo, las de un brazo robótico. Simscape Fluids; nos permite modelar elementos que hagan uso de la mecánica de fluidos. Contando con la ventaja de poderlos integrar con otros tipos de sistemas que dispone Simulink®, la figura 8 muestra un esquema de los componentes que maneja cada paquete (Mathworks®, Simscape, s.f.).

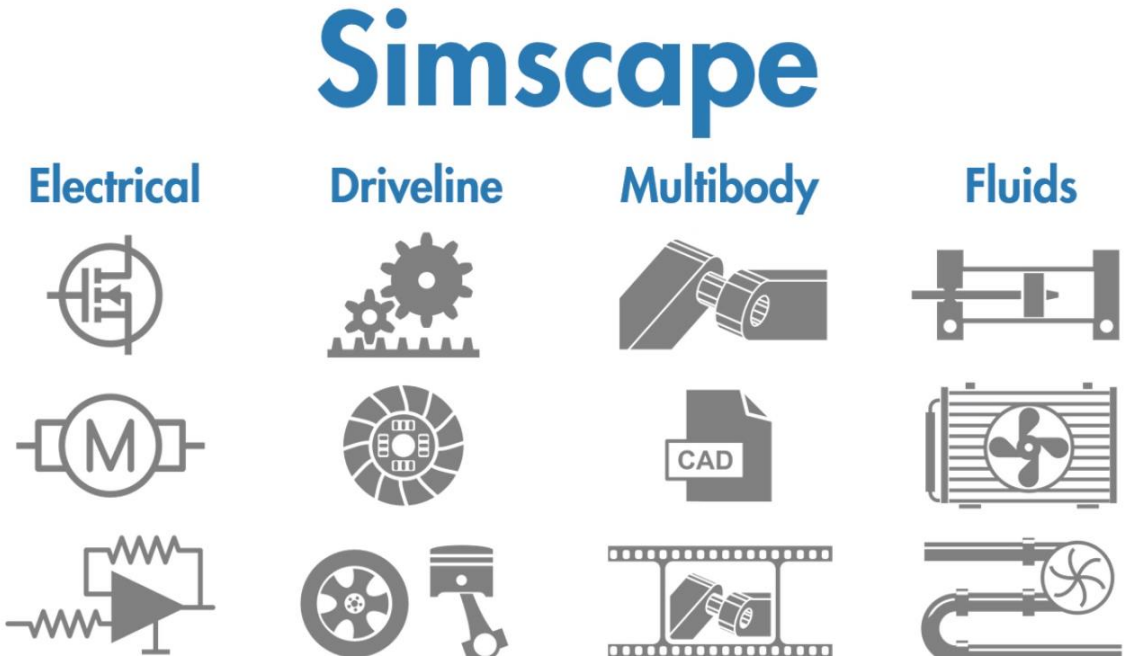


Figure 8. Paquete Simscape™ (Mathworks®, Simscape, s.f.).

Simscape ayuda a desarrollar sistemas de control y probar el rendimiento a nivel del sistema. Puede crear modelos de componentes personalizados utilizando el lenguaje Simscape basado en MATLAB®, que permite la creación basada en texto de componentes de modelado físico, dominios y bibliotecas. Puede parametrizar sus modelos utilizando las variables y expresiones de MATLAB, y diseñar sistemas de control para su sistema físico en Simulink. Para implementar sus modelos en otros entornos de simulación, incluidos los sistemas de hardware en bucle (HIL), Simscape admite la generación de código C (Mathworks®, Simscape, s.f.).

1.5 CONFIGURACIÓN CINEMÁTICA DEL ROBOT

En los robots existen múltiples juntas, que nos permiten realizar diferentes movimientos, la ubicación de los mismos con su respectivo eslabón permite adaptar el espacio de la tarea de un brazo robótico a las necesidades requeridas, los ejes de un robot se distinguen de las siguientes maneras:

Eje Rotativo: Es un ensamble entre dos miembros rígidos el cual permite que uno gire en relación con el otro alrededor de un eje fijo.

Eje traslativo: Es un ensamble entre dos miembros rígidos que permite que uno tenga movimiento lineal en contacto con el otro.

Junta compleja: Es un ensamble entre dos miembros rígidos estrechamente relacionados que permiten que uno gire con respecto al otro alrededor de un eje móvil.

La cadena cinemática se puede combinar mediante ejes traslativos y rotatorios. La variedad de posibles variaciones de una estructura de robot industrial se puede determinar de la siguiente manera, como se indica en la figura 9 y 10 **(Y. Nof, March 1999)**.

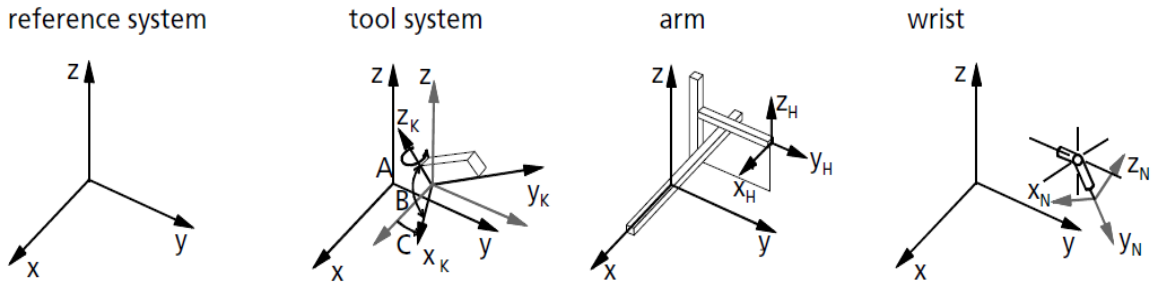


Figure 9. Definición de sistemas de coordenadas para la tarea de manipulación y el robot (Y. Nof, March 1999).

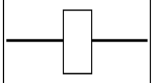
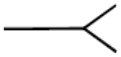
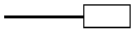
System	Translatory axis		Rotary axis		Gripper	Tool	Separation of arm and wrist
	telescopic	traverse	pivot	hinge			
Symbol							

Figure 10. Símbolos para la descripción de la estructura cinemática de robots industriales según la directriz VDI 2681 (Y. Nof, March 1999).

La figura 11, muestra los 6 tipos de robots industriales más comunes, con su respectiva área de trabajo, tipo de eje y su esquema cinemático, donde DOF es el número de grados de libertad. Los robots industriales normalmente tienen cuatro grados de libertad en los ejes principales, Si el número de ejes independientes (brazo y muñeca) es mayor que seis, hablamos robots cinemática mente redundantes. Debido a que hay más juntas que el número mínimo requerido, pueden existir movimientos internos que permiten que el manipulador se mueva mientras se mantiene la posición del efector final fija (Y. Nof, March 1999).

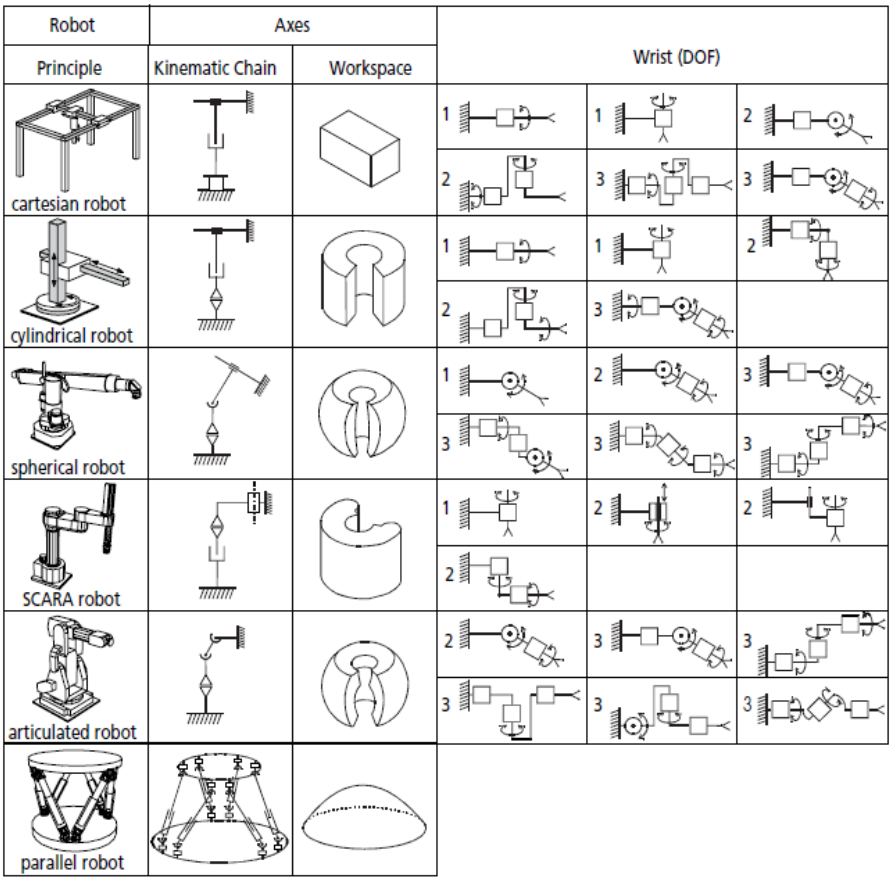


Figure 11. Configuraciones típicas de brazos y muñecas de robots industriales ***(Y. Nof, March 1999).***

1.6 CINEMÁTICA DEL ROBOT

Los modelos geométricos y cinemáticos del robot se emplean tanto para la simulación como el control, se estudia el modelo directo que permite obtener la posición y orientación del efector final en función de las variables articulares. El modelo inverso, el cual obtiene las variables articulares que hacen que la posición y orientación del efector final sea la deseada (Ollero Baturone, 2005) (Hassan & Abomoharam, 2017).

1.6.1 El Problema Cinemático Directo

El problema cinemático directo viene dado por una función que permite expresar la posición y orientación del sistema de referencia objetivo en el espacio cartesiano, p en términos de las variables articules q, Siendo φ un conjunto de funciones no lineales (Baturone, 2005) (Milanés Hermosilla & Castilla Pérez, 2016).

$$p = \varphi(q) \quad \textit{Ec. 1}$$

La obtención del modelo cinemático directo puede ser abordado mediante dos enfoques diferentes denominados métodos geométricos: son adecuados para casos simples, su aplicación queda limitada a robots con pocos grados de libertad; Los métodos basados en cambios de sistemas de referencia (Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, 2007).

1.6.2 El Problema Cinemático Inverso

El objetivo del problema cinemático inverso consiste en encontrar los valores que deben adoptar las coordenadas articulares del robot, para que su extremo se posicione y oriente según una determinada localización espacial (Izaguirre, Hernández, Rubio, Prieto, & Hernández, 2011). Existen diversos métodos de resolución del problema cinemático inverso para el caso de un máximo de 3 grados de libertad, métodos geométricos, matriz de transformación homogénea, y para el caso de más de 3 grados de libertad el desacoplo cinemático (Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, Fundamentos de robotica, 2007) (Milanés Hermosilla & Castilla Pérez, 2016).

1.7 DINÁMICA DEL ROBOT

El modelo dinámico del robot manipulador permite explicar todos los fenómenos físicos que se encuentran en su estructura mecánica, tales como efectos inerciales, fuerzas centrípetas y de coriolis, par gravitacional y fricción los cuales son fenómenos físicos intrínsecos o propios de la naturaleza dinámica del robot (Reyes Cortes, Robotica Control de Robots Manipuladores, 2011). También el modelado dinámico es esencial para la simulación, el control y la optimización del diseño de los manipuladores (Arian, Danaei, Abdi, & Nahavandi, 2017).

1.7.1 Modelo dinámico de la estructura mecánica de un robot rígido

El modelo dinámico representa la base matemática para llevar a cabo el análisis y estudio de los fenómenos físicos presentes en la estructura mecánica de un robot manipulador de n grados de libertad en cadena cinemática abierta (Para sistema robótico de cadena cerrada debe convertirse virtualmente a uno cinemático abierto), con eslabones rígidos, en su rango de operación nominal o ancho de banda. Para el caso que el robot tenga eslabones flexibles, es necesario incorporar dentro del modelo dinámico, el fenómeno físico de elasticidad y flexibilidad en los eslabones (Reyes Cortes, Robotica Control de Robots Manipuladores, 2011) (Shafei & Shafei, 2018).

1.7.2 Obtención del modelo dinámico de un robot mediante la formulación de lagrange-euler

Para la obtención de un modelo dinámico de un robot manipulador de n grados de libertad, formado por eslabones rígidos conectados por articulaciones libres de elasticidad en cadena cinemática abierta, existen diversas herramientas matemáticas, un método estándar para obtener el modelo dinámico es mediante las ecuaciones de movimiento de Euler-Lagrange (Reyes Cortes, Robotica Control de Robots Manipuladores, 2011). Tratándose de un procedimiento ineficiente desde el punto de vista computacional, puede comprobarse que el algoritmo es de un orden de complejidad computacional O(n^4), es decir, el número de operaciones a realizar crece con la potencia 4 del número de grados de libertad. Sin embargo, conduce a unas ecuaciones finales bien estructuradas donde aparecen de manera clara los diversos pares y fuerzas que intervienen en el movimiento (inercia, coriolis, gravedad) (Hassan & Abomoharam, 2017) (Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, Fundamentos de robotica, 2007) (Sciavicco, Siciliano, & Villani, 1994).

1.8 CONTROL CINEMÁTICO

Partiendo de las especificaciones de movimiento requeridas, el control cinemático se encarga de obtener las trayectorias que debe seguir cada articulación del robot a lo largo del tiempo para lograr los objetivos fijados por el usuario (punto de destino, trayectoria cartesiana del efector final del robot, tiempo invertido por el usuario, etc.) (Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, Fundamentos de robótia, 2007) (Renteria & Rivas, 2000).

1.8.1 Funciones del control cinemático

La Figura 1 muestra, de manera esquemática, el funcionamiento del control cinemático. Recibe como entradas los datos de movimiento procedentes del programa del robot escrito por el usuario (punto de destino, precisión, tipo de trayectoria deseada, velocidad o tiempo invertido, etc.) (Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, Fundamentos de robótia, 2007). Posteriormente se debe disponer del modelo cinemático del robot que permite conocer las relaciones entre: Las posiciones del extremo del robot, expresadas en el espacio cartesiano, y las coordenadas articulares del robot. Las velocidades de movimiento del extremo del robot y las velocidades en las articulaciones (Rentería & Rivas, 200) (Restrepo, Villegas, Arias, Sergio, & Madrigal, 2012) (Acosta Sánchez & Sigut Saavedra, 2005).

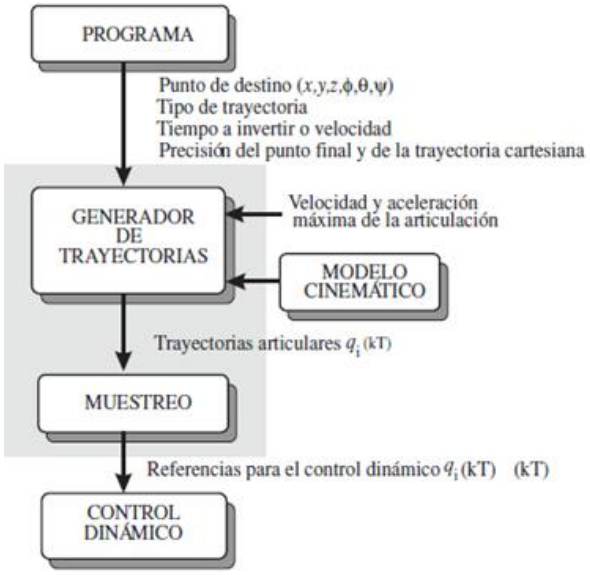


Figure 12. Funcionamiento del control cinemático ***(Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, Fundamentos de robótia, 2007).***

1.8.2 Tipos de trayectorias

Para realizar una tarea determinada el robot debe moverse desde un punto inicial a un punto final. Este movimiento podría ser realizado según infinitas trayectorias espaciales. De todas ellas hay algunas que, bien por su sencillez de implementación o bien por su utilidad y aplicación a diversas tareas, son las que en la práctica incorporan los robots comerciales, siendo su generación función del control cinemático. De este modo, puede encontrarse que los robots dispongan de trayectorias punto a punto (Movimiento eje a eje, Movimiento simultáneos de ejes, Trayectorias coordinadas o isócronas), y trayectorias continuas (Milanés Hermosilla & Castilla Pérez, 2016) (Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, Fundamentos de robótia, 2007) (Acosta Sánchez & Sigut Saavedra, 2005).

1.8.3 Generación de trayectorias cartesianas

Generalmente, el usuario indica el movimiento que el robot debe realizar especificando las localizaciones por las que debe pasar el efector final del robot, junto con otros datos, como instantes de paso, velocidades o tipos de trayectorias. En ocasiones, el usuario debe especificar una secuencia de posiciones por las que se desea que pase el efector final del robot, siendo preciso establecer un interpolador entre las localizaciones expresadas en el espacio de la tarea (Flórez Vergara, Castro Riveros, & Castillo Estepa , 2017) (Espinosa Zapata, Mazo, López Guillén, & Ureña, 1998).

1.8.3.1 Evolución de la orientación

Como es conocido, la especificación de la posición en el espacio de la tarea se realiza habitualmente, y salvo escasas excepciones, en coordenadas cartesianas. Sin embargo, la especificación de la orientación puede realizarse mediante diferentes herramientas, como son: matrices de rotación, ángulos de Euler (en sus diferentes versiones), par de rotación o cuaternios. (Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, Fundamentos de robótica, 2007)

1.8.4 Interpolación de trayectorias

Una vez que se dispone de la secuencia de configuraciones articulares por las que debe pasar el robot, la siguiente función del control cinemático, es la de unir esta sucesión de puntos articulares garantizando, junto con las condiciones de configuración-tiempo de paso, que se cumplan las restricciones de velocidad y aceleración máximas asociadas a los actuadores del robot, de manera que se consiga una trayectoria realizable y suficientemente suave (Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, Fundamentos de robótica, 2007) (Suñer, Valero, Ródenas, & Besa, 2007).

Las funciones interpoladoras utilizadas con mayor frecuencia son Interpoladores lineales, Interpolador splin cúbico, Interpolador splin quíntico, Interpoladores trapezoidales. Cada una de ellas ha sido desarrollada para un solo grado de libertad, debiendo quedar claro que el mismo cálculo deberá repetirse para cada uno de los grados de libertad del robot. Cabe indicar que si bien las técnicas de interpolación que se nombran, están planteadas para el espacio articular son igualmente aplicables para el espacio de la tarea (Soto Cajiga, Vargas Soto, & Pedraza Ortega, 2006) (Barrientos Cruz A. , Balaguer, Bala, Peñin, & Aracil, Fundamentos de robótica, 2007).

1.10 ESTADO DEL ARTE

Tabla 1
Estado del Arte

IDENTIFICACIÓN	OBJETIVO GENERAL	CATEGORÍAS/ VARIABLES	INSTRUMENTOS RECOLECCIÓN DE LA INFORMACIÓN	RESULTADOS
Yiqun Zhou, Jianjun Luo, Mingming Wang. Dynamic coupling analysis of multi-arm space robot. Acta Astronáutica. Volume 160, July 2019, Pages 583-593	analizar el efecto de acoplamiento dinámico del robot espacial con múltiples brazos de bucle abierto y el sistema de bucle cerrado.	Robot espacial de brazos múltiples. Sistema de circuito cerrado. Factor de acoplamiento dinámico. Acoplamiento dinámico elipsoide Dinámica. Cinemática.	Implementación del modelado cinemático basado en la teoría del vector espacial y la teoría de la gráfica. Los factores de acoplamiento dinámico y las matrices de acoplamiento dinámico e proponen como dos herramientas para medir el grado de acoplamiento	Las dos herramientas se aplican para analizar el acoplamiento de un robot espacial de doble brazo y su cuerpo compuesto con el objetivo capturado, donde se estudian principalmente las influencias de los parámetros dinámicos, las longitudes de enlace y las variables conjuntas. (Zhou, Luo, & Wang, 2019)
Erol Özgür, Youcef Mezouar. Kinematic modeling and control of a robot arm using unit dual quaternions. Robotics and Autonomous Systems. Volume 77, March 2016, Pages	formular la cinemática de movimiento (posición + velocidad), como el control de posición de un brazo robótico de n GDL de una manera eficiente, recurriendo a la teoría de screw	Cuaternios dobles. Brazo robótico. Cinemática. Controlar	Uso de la unidad de cuaternios dobles y su álgebra. Modelado cinemático y control de posición de brazos robóticos múltiples grados de libertad.	La formulación presentada se valida en un brazo robot Kuka LWR IV de 7 GDL, tomando una botella que está sobre una mesa desde una postura conocida, luego, se corrige la postura de la botella y la vuelve a colocar. El modelado es compacto y rápido. Por lo tanto, el cálculo de la ley de control es rápido.

66-73	expresada a través de cuaternios dobles y su álgebra.			(Özgü & Mezouar, 2016)
Gang Xiong Ye Ding LiMin Zhu. Stiffness-based pose optimization of an industrial robot for five-axis milling. Robotics and Computer-Integrated Manufacturing. Volume 55, Part A, February 2019, Pages 19-28	Presentar un método de optimización de posición para el robot de fresado al convertir una trayectoria de herramienta CNC de cinco ejes en una trayectoria de robot industrial de seis ejes comercial.	Mecanizado de robots. Fresado de cinco ejes. Índice de rendimiento de rigidez. Eliminación de redundancia. Posicionar la optimización. Restricciones cinemáticas	Se presenta un método de optimización de posición para eliminar la libertad redundante a lo largo de una trayectoria de herramienta CNC de 5 ejes dada con el objetivo de aumentar la rigidez del robot, lo que resulta en una mayor precisión de mecanizado. Se propone un nuevo índice de rendimiento invariante del cuadro para evaluar la rigidez del robot en una postura determinada. El problema de la eliminación de la redundancia se formula como un problema de optimización unidimensional teniendo en cuenta las limitaciones de los límites comunes, la evitación de la singularidad y la suavidad de la trayectoria. El problema de la optimización unidimensional se resuelve mediante un algoritmo de búsqueda de desratización simple.	el índice de rendimiento y el algoritmo de optimización de la trayectoria del robot se validan mediante simulaciones y experimentos en un robot industrial, demostrando que la precisión de mecanizado se puede mejorar de manera eficiente con el método propuesto. (Xiong, Din, & Zhu, 2019)

Alaleh Arian, Behzad Danaei, Hamid Abdi, Saeid Nahavandi. Kinematic and dynamic analysis of the Gantry-Tau, a 3-DoF translational parallel manipulator. Applied Mathematical Modelling. Volume 51, November 2017, Pages 217-231	Implementar el modelado matemático de la cinemática y la dinámica del manipulador Gantry-Tau de 3 grados de libertad.	Análisis cinemático. Análisis dinámico. Manipulador paralelo. Newton – Euler. Trabajo virtual Singularidad; posición. Velocidad y aceleración	Plataforma robot paralelo Gantry-Tau. Herramientas matemáticas como, cinemática inversa, jacobino, modelo dinámico inverso con Enfoque de trabajo virtual y Newton-Euler. Modelo dinámico en SimMechanics	se simula un perfil de trayectoria y el perfil de torque calculado utilizando los dos métodos se compara con un modelo de SimMechanics para verificar que produzcan la salida correcta. (Arian, Danaei, Abdi, & Nahavandi, 2017)
Alaa Hassan, Mouhammad Abomoharam. Modeling and design optimization of a robot gripper mechanism. Robotics and Computer-Integrated Manufacturing. Volume 46, August 2017, Pages 94-103	realizar un estudio detallado de la pinza del robots, para ilustrar las interacciones entre sus pasos, y determinar la fuerza óptima extraída por la pinza sobre la superficie.	Modelado de robots. Optimización de diseño multicriterio. NSGA-II. Análisis de sensibilidad	se establece el modelo geométrico para encontrar la relación entre las coordenadas operacionales. Se deriva una matriz jacobiana equivalente para encontrar el modelo cinemático. el modelo dinámico se obtiene utilizando la formulación de Lagrange-Euler. El diseño del dispositivo de agarre se resuelve utilizando un algoritmo genético de clasificación no dominado versión II (NSGA-II).	se presenta un estudio analítico paso a paso para encontrar modelos geométricos, cinemáticos y dinámicos. La etapa de optimización muestra el procedimiento para formular y resolver un problema de optimización del diseño y luego analizar la sensibilidad del diseño. (Hassan & Abomoharam, 2017)

Hui Zhang, Hongzhe Jin, Zhangxing Liu, Yubin Liu, Yanhe Zhu, Jie Zhao. Real-time Kinematic Control for Redundant Manipulators in a Time-varying Environment: Multiple-dynamic Obstacle Avoidance and Fast Tracking of a Moving Object. IEEE Transactions on Industrial Informatics. Volume: 4, Issue: 2, April 2019,	Aplicar una estrategia de control cinemático en tiempo real para realizar un seguimiento rápido de los manipuladores redundantes en el entorno de cambio de tiempo.	Evitación de obstáculos de robots. Control cinemático de robots redundantes. PID. Red neuronal. Manipuladores. redundantes.	Se propone un método de evitación de obstáculos basado en LCE, para ajustar los movimientos del manipulador en tiempo real, define la energía total para el efector final. En la planificación en tiempo real, se eleva un modelo PID de neurona única sin supervisión para aumentar de forma adaptativa, la relación de convergencia del seguimiento mediante el aprendizaje en línea de PCA. combinado con el método de evitación dinámica de obstáculos propuesto, se establece una estrategia de control cinemático para manipuladores redundantes	Implementada la estrategia de control cinemático para manipuladores redundantes, se rastre un objeto en movimiento rápidamente en un entorno variable en el tiempo. El análisis teórico y los resultados experimentales, verifican la viabilidad y la rápida convergencia de la estrategia propuesta. (Zhang, y otros, 2019)
Dimitrios Papageorgiou, Theodora Kastritsi, Zoe Doulgeri. A passive robot controller aiding human coaching for kinematic behavior modifications Robotics and	Implementar un esquema de control mediante háptica utilizando los parámetros de la dinámica de movimiento de un brazo robótico.	Enseñanza cinestética. Comportamiento cinemático. Dinámicas de Movimiento. Trayectorias. Cinemático. Háptica. Cuaternarios. Dinámica.	Se propone un controlador para la modificación cinestésica de un comportamiento cinemático codificado por las Dinámicas de Movimiento Dinámico (DMP). El controlador permite al profesor humano "inspeccionar" de forma háptica las propiedades	Los resultados experimentales demuestran la comunicación bidireccional establecida por el esquema propuesto, y el tiempo significativamente reducido que se logra en una tarea de enseñanza al robot.

Computer-Integrated Manufacturing Volume 61, February 2020.		Control.	espaciales del comportamiento aprendido en SE (3). Se realizan experimentos sobre la enseñanza de una variante de una tarea de fresado emulada con un manipulador KUKA LWR4 +.	Implica una reducción de carga cognitiva para el usuario en comparación con la enseñanza del robot con compensación de gravedad, que no tiene en cuenta ningún conocimiento del conjunto de datos de entrenamiento anterior. (Papageorgiou, Kastritsi, & Doulgeri, 2020)

Fuentes obtenidas de artículos científicos

CAPÍTULO 2, ESTRUCTURA MECÁNICA Y MODELO DINAMICO DEL ROBOT

En esta sección se explica la elaboración del diseño de la estructura del robot, y la generación de la mecánica del cuerpo mediante simscape, su experimentación con respectivo análisis y los inconvenientes que se presentaron.

2.1 DISEÑO DEL ROBOT

En este apartado se muestra el diseño de la estructura del robot realizado en software Solidworks®, en los siguientes incisos se muestra el proceso de planeación, elaboración y los inconvenientes que se presentaron durante el diseño.

2.1.1 Planeación

Durante los primeros días de la pasantía de investigación, se tomaron ciertas pautas en el diseño las cuales son; los motores se ubican la base, esto con el fin de evitar cargas adicionales en cada eslabón del robot y a si mejorar la capacidad de carga del mismo. La mayoría de partes del robot se puedan mecanizar mediante corte laser y solo ciertas piezas se realicen en impresión 3D, esto con el fin de optimizar tiempo y costos, ya que la impresión 3D es bastante tardada. Tener un coeficiente de fricción bastante bajo en las juntas de unión de los eslabones, para esto se optó colocar rodamientos ya que nos permite minimizar en gran medida los problemas de fricción y desgaste de las juntas otorgando un movimiento suave.

Teniendo estas pautas claras se procede a investigar estructuras que se puedan adaptar a estas caracterizas de forma fácil, tomando la decisión se selecciona como referencia un robot tipo MeArm específicamente este diseño se basa el robot Dobot® v1.0 (dobot®, s.f.). Se toma este diseño como referencia ya que esta estructura cuenta con la gran ventaja de contar con eslabón que no requiere de un actuador, esto con un mecanismo que mediante el movimiento del brazo y antebrazo siempre mantiene el eslabón del efector final paralelo a la base, en pocas palabras, este es un robot paralelo antropomórfico, además la carga de los actuadores de este diseño está en el hombro, y hay que mencionar que este eslabón solo gira paralelo a la base.

2.1.2 Elaboración del diseño en el CAD

En esta etapa teniendo claro el tipo de robot que se basa este proyecto, se procede a elaboración tomando en cuenta las características expuestas anteriormente. En la figura 13 se muestra el resultado final. Las siguientes figuras indica las piezas que conforma cada parte del robot, como son; base, hombro, brazo, antebrazo, y efector final.

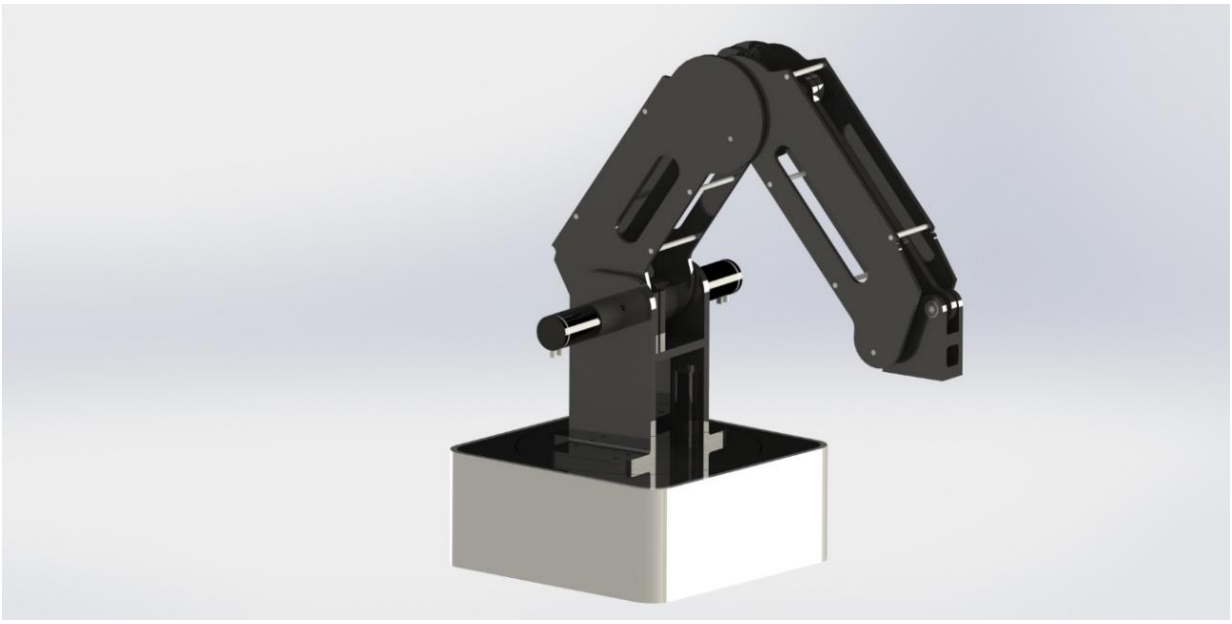


Figure 13. Render del brazo robótico (Solidworks®).

Se toma la decisión de diseñar una base en forma de caja (figura 14) donde se pueda almacenar los controladores de los actuadores y la tarjeta Arduino® Mega, teniendo en cuenta la estética y la ubicación de los pines de conexión del mismo. Esta caja de control por su diseño de esquina redondeadas se tiene que mecanizar en impresión 3D.

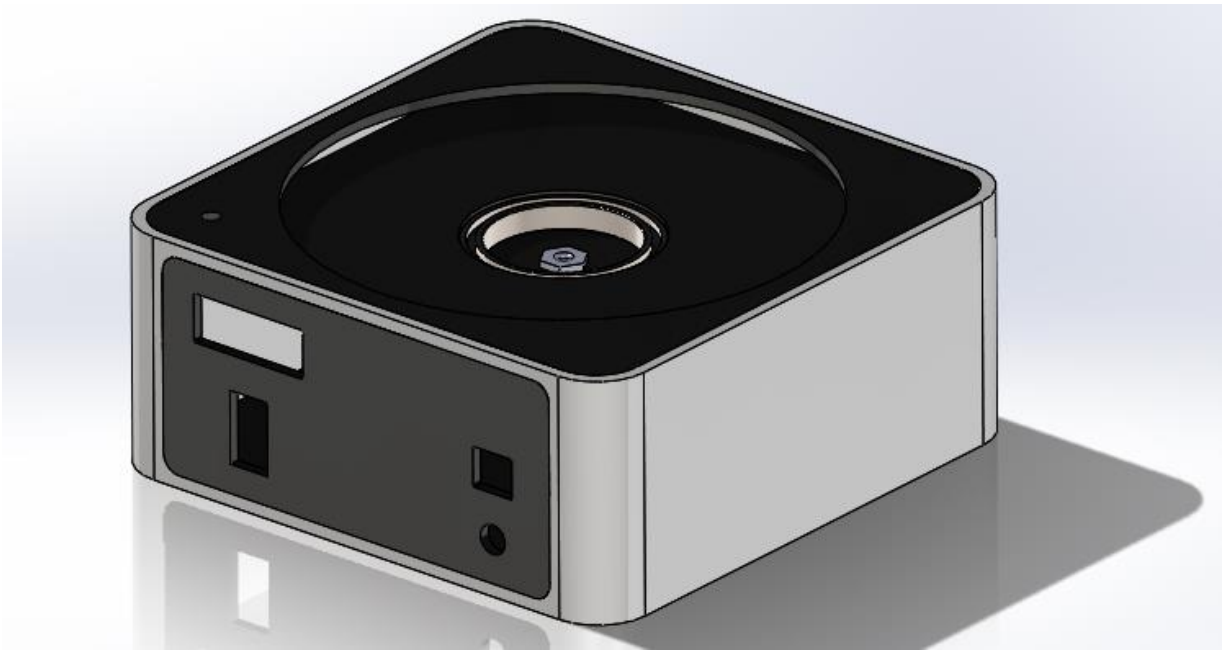


Figure 14. Base del robot.

En la figura 15, se muestra una vista de los componentes que conforman la base, para dar una mejor perspectiva del mismo. El rodamiento que se usa en la base es de tipo blindado de la referencia 6808zz con las medidas 5.2x4x0.7 cm, contando con la ventaja de tener dimensiones reducidas y así optimizar el espacio, es capaz de operar en altas revoluciones a 10k RPM sin grasa y 12k RPM con grasa, con precisión A1 de materia acero cromado, además de ser muy económico (AST, s.f.). características importantes para su selección en este proyecto.

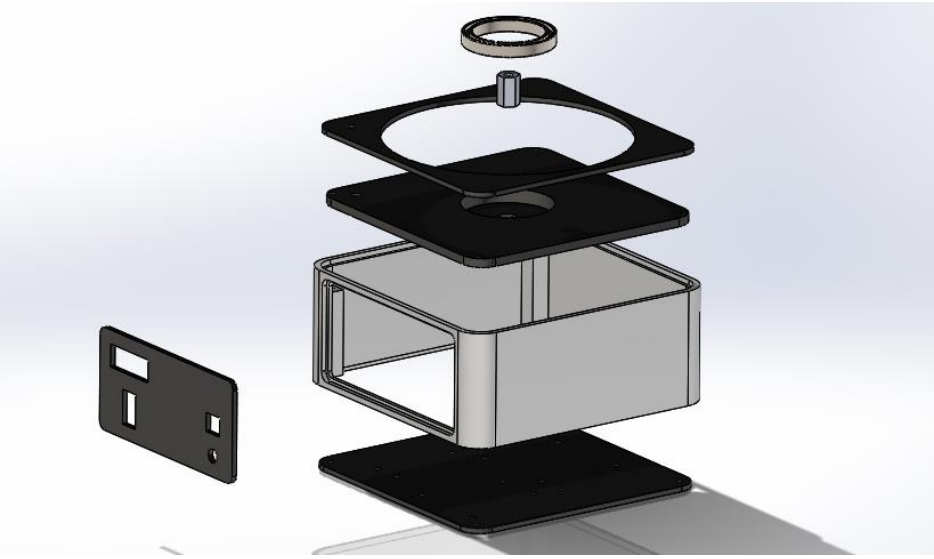


Figure 15. Base del robot, vista explosionada.

En la figura 16, se muestra la parte del hombro de este robot, como se observa este eslabón contiene los actuadores evitando la carga en los demás eslabones y del mismo, ya que este solo rota en el eje z del mismo.

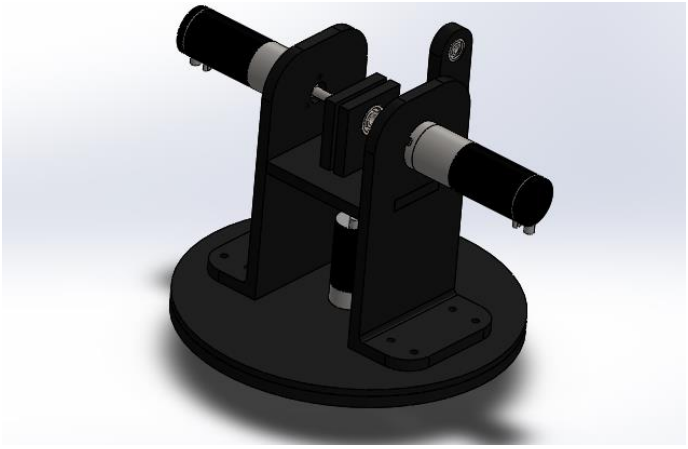


Figure 16. Hombro del robot

Como se observa en la figura 17, cada articulación cuenta con su rodamiento blindado. La referencia son 623zz con las medidas 3x10x4 Mm, lo que nos indica que son muy compactos, esta es la principal razón para su selección, no requieren mantenimiento lo que a largo plazo es bastante conveniente.

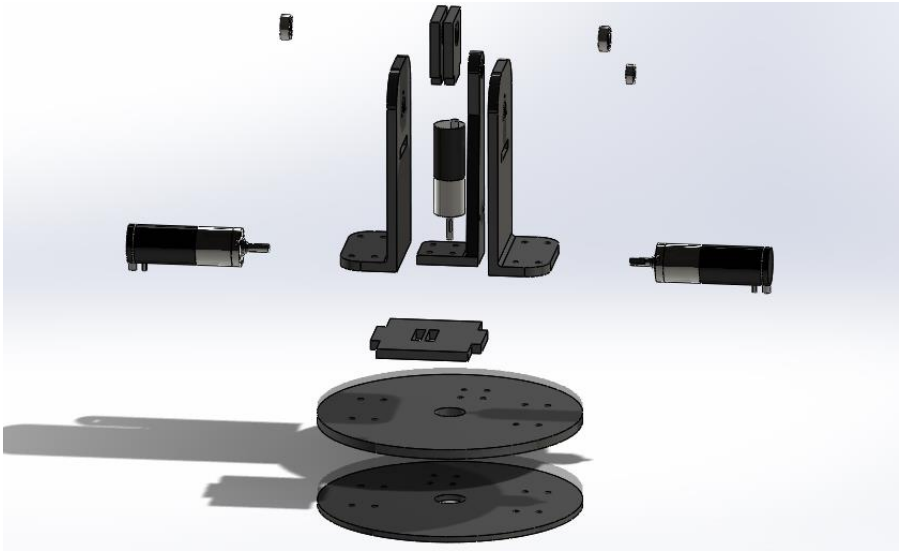


Figure 17. Hombro del robot, vista explosionada.

El eslabón del brazo (figura 18) dispone de una pieza de acoplamiento hexagonal con sujeción con tornillo prisionero para sujetar el eje del actuador, cuenta con un orificio en cada lado con el fin de aligerar la estructura, los pequeños ejes que sujetan cada extremo tienen la tarea de brindar rigidez a la estructura.

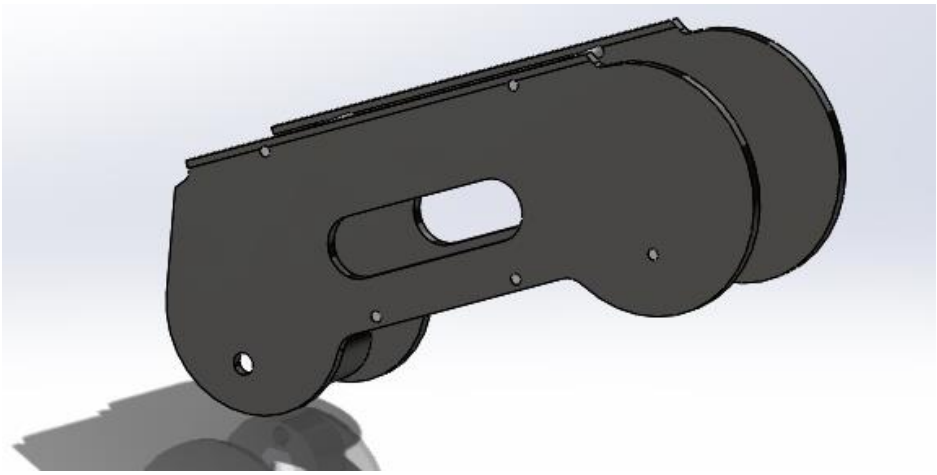


Figure 18. Brazo del robot

Este eslabón tiene el diseño más simple ya que cuenta con pocas piezas.

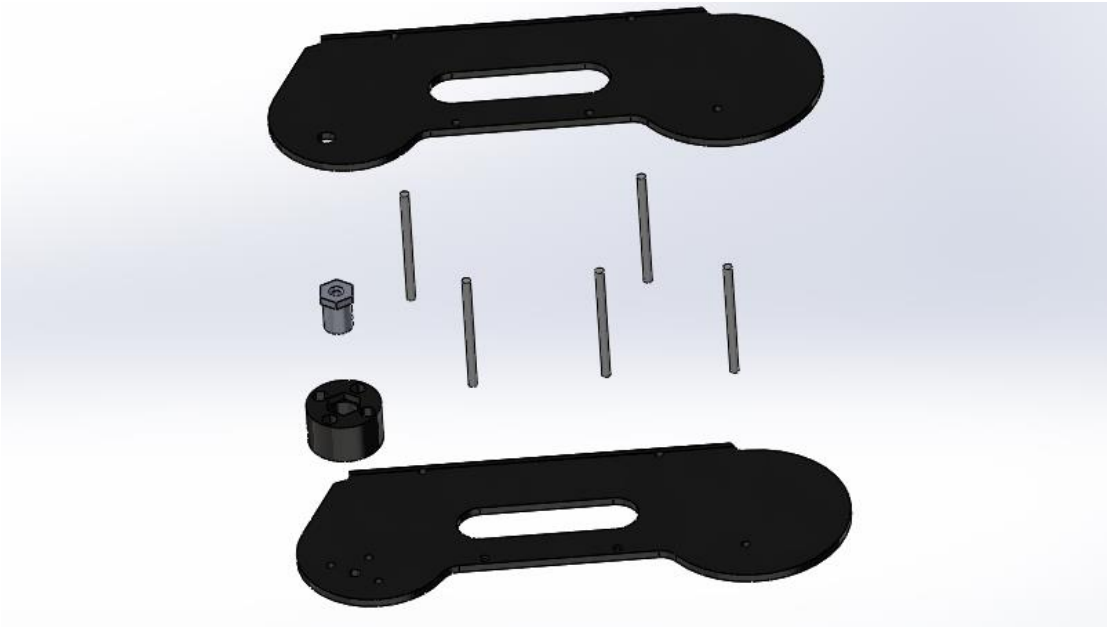


Figure 19. Brazo del robot, vista explosionada.

El antebrazo cuenta con dos barras que permite la transmisión del movimiento del actuador, como vemos en la figura 20, uno de los eslabones cuenta con acoplamiento hexagonal con tornillo prisionero para sujetar el eje del motor, los rodamientos compactos que dispone son de la referencia 623zz mencionado anteriormente.

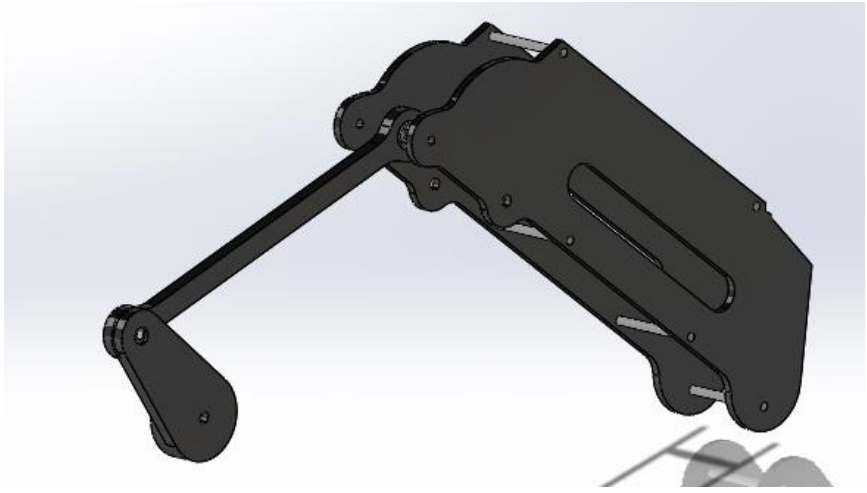


Figure 20. Antebrazo del robot y mecanismo de transmisión.

Al igual que el eslabón anterior, el antebrazo cuenta con unas pequeñas barras que brinda la

rigidez suficiente a la estructura, como se visualiza en la vista explosionada de la figura 21.

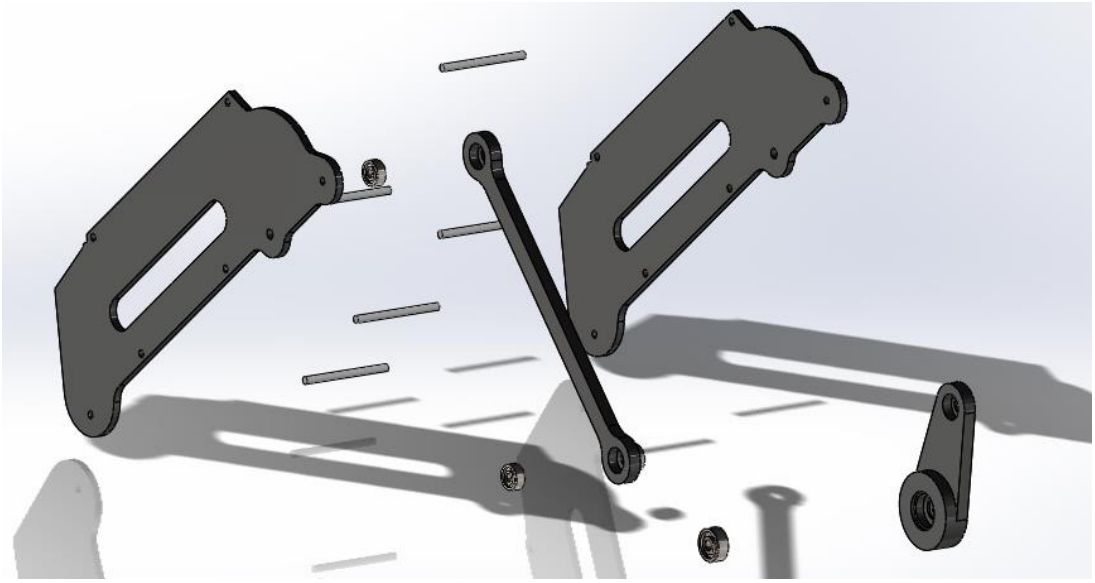


Figure 21. Antebrazo del robot, vista explosionada.

. Se muestra en la figura 22, el mecanismo que permite mantener el efector final siempre en paralelo independiente del movimiento del brazo y antebrazo, cuenta con dos barras (L1 y L3) y un eslabón intermedio (L2) el cual su centro está en la unión del brazo y antebrazo. La barra L1 va sujeta mediante una junta a otra barra fija (P) en el codo, la otra barra (L3) va sujeta mediante una junta al efector final, cada barra cuenta con sus respectivos rodamientos 623zz en las juntas.

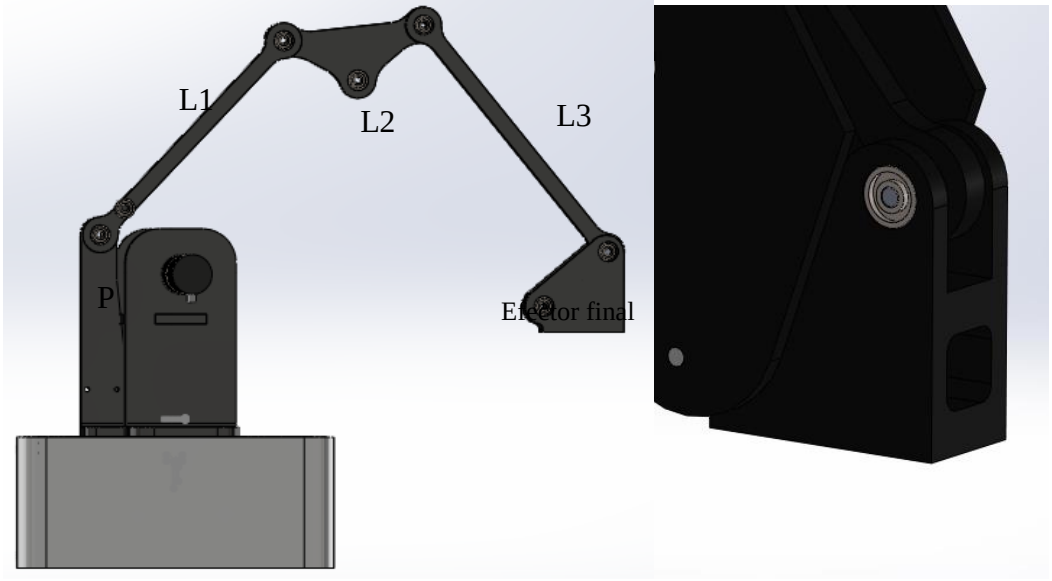


Figure 22. Mecanismo del efector final.

El efector final cuenta con pequeño orificio que se sujeta con un tornillo prisionero, donde se puede insertar diferentes tipos de herramientas como una pinza, un láser, una ventosa, etc. La figura 23, muestra una vista explosionada de este mecanismo para una mejor interpretación.

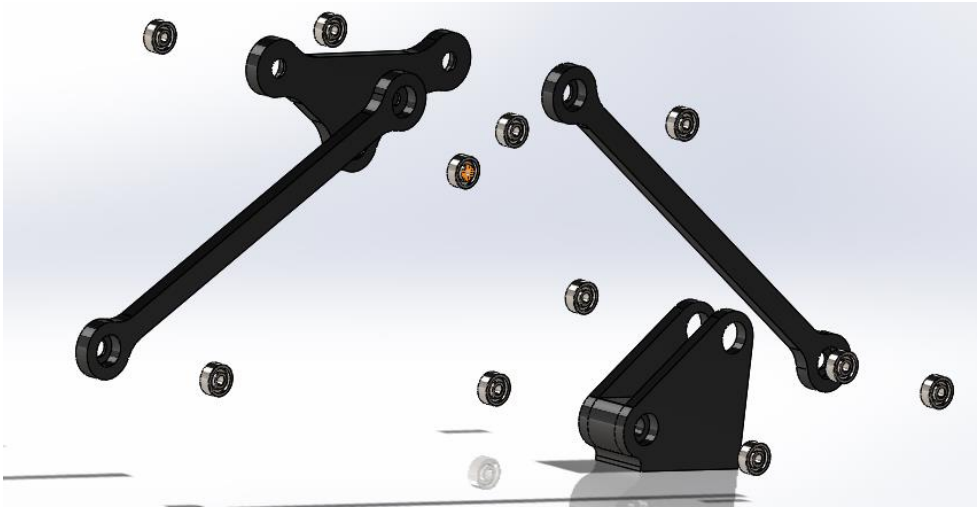


Figure 23. Mecanismo del efector final, vista explosionada.

2.2 Modelo en Simscape Multibody, (Simulink®).

Para exportar un modelo CAD a Simscape Multibody se debe instalar y habilitar el plugin en el CAD que en este caso se usa Solidworks®, una vez se exporta, se obtiene el siguiente diagrama de bloques en Simulink® de la figura 24.

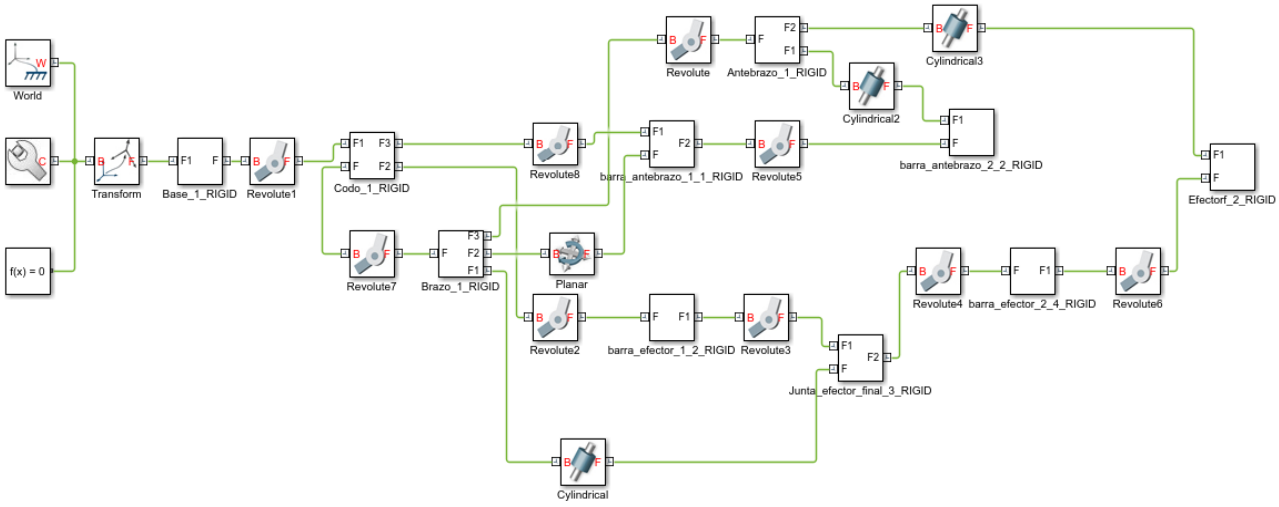


Figure 24. Diagrama de la estructura del robot en Simscape (Simulink®).

Esta herramienta permite obtener el modelo dinámico del robot de una forma más intuitiva y rápida. Los tres primeros bloques definen el entorno de la simulación mecánica de sistema, la gravedad y ajustes de solución usados en la simulación, en la figura 25, se muestra los menús de cada bloque, permitiendo modificar los parámetros mencionados.

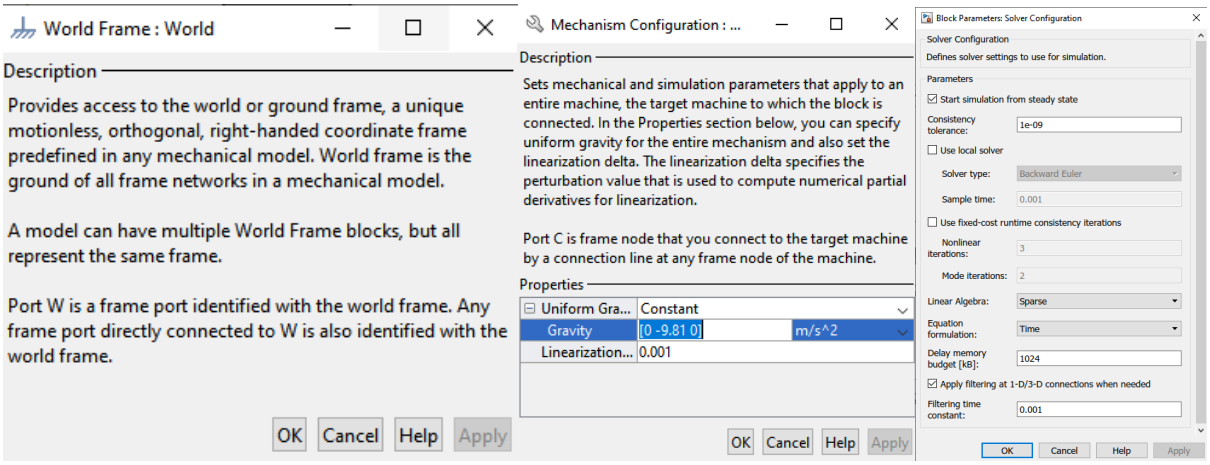


Figure 25. Descripción de los bloques world, mechanism configuration y solver configuratión.

La figura 26, muestra el brazo robótico en el torno mechanics explorer (entorno de simulación) en Matlab®, dispone de opciones como las de reproducción, general videos, cambien el color de fondo, ver sus centros y otras más.

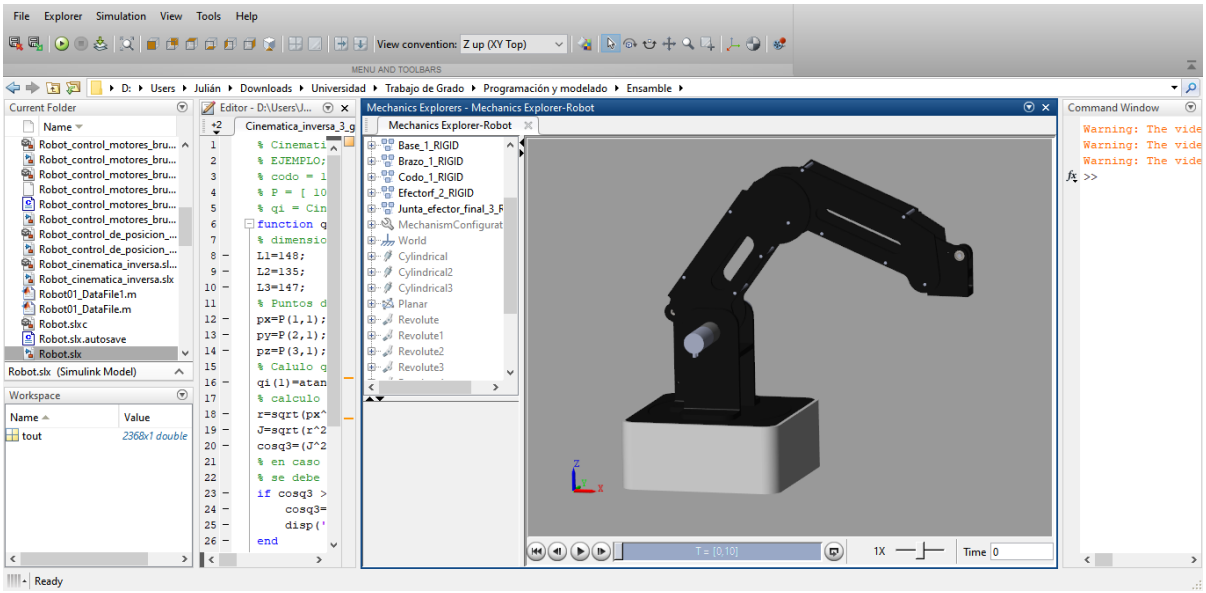


Figure 26. brazo robótico en Simscape.

Las gráficas 27 y 28, indican el comportamiento de posición y velocidad de cada uno de los eslabones sin ningún tipo de control, bajo el efecto de la gravedad y un cálculo automático del movimiento de las juntas, durante un tiempo de 0 a 10 segundos, en resumen, se ejecuta el diagrama de la figura 24, sin ningún tipo de control y se muestra el resultado obtenido.

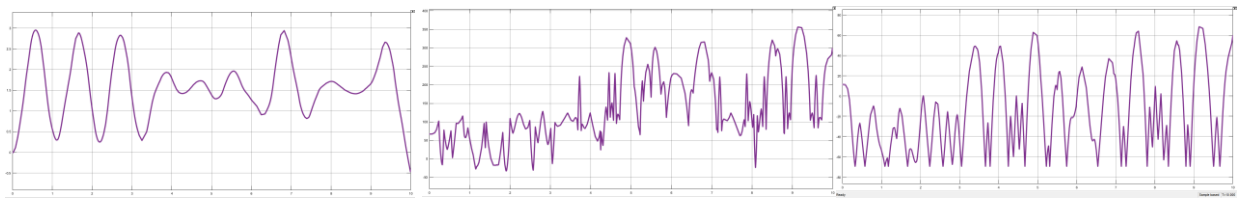


Figure 27. Valores de posición en rad, de hombro, brazo, antebrazo, generación de moviente automático bajo efecto de la gravedad en un tiempo de 10 segundos.

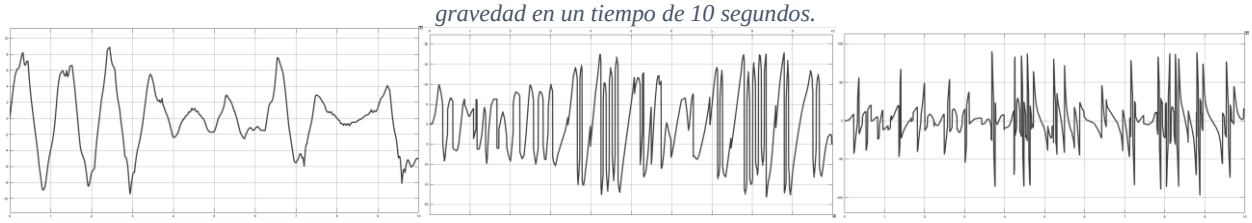


Figure 28. Valores de velocidad angular en rad/s, de hombro, brazo, antebrazo, generación de moviente automático bajo efecto de la gravedad en un tiempo de 10 segundos.

Como vemos en las gráficas 27 y 28, el movimiento que se genera es de una forma totalmente descontrolada, en el entorno de simulación lo que se visualiza es un choque entre eslabones con movimientos erráticos, lo que demuestra lo bien que funciona esta herramienta, ya que no permite mover la articulación más allá del límite físico. A continuación, se realiza un control de movimiento, ingresando una trayectoria generada con el bloque de signal builder a las juntas del hombro, brazo y antebrazo como se muestra en la figura 29 (diagrama ordenado por eslabones), simulando a un actuador.

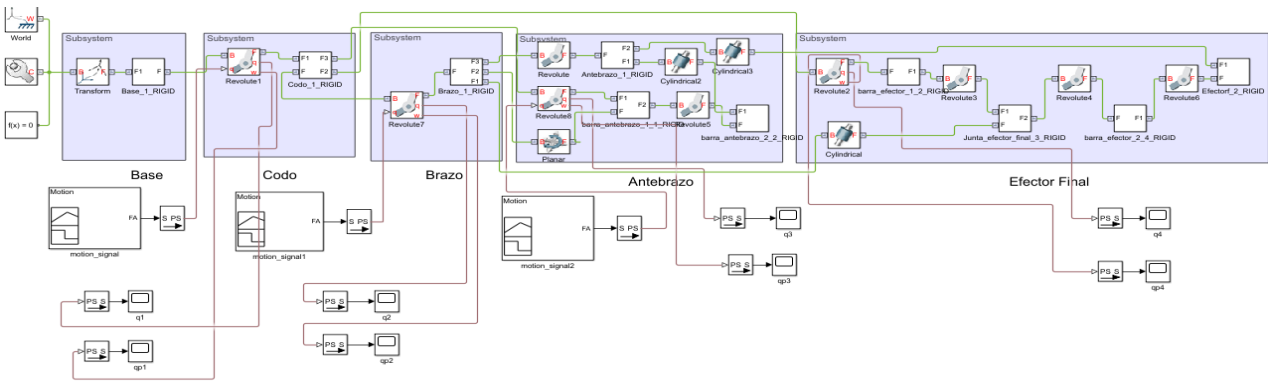


Figure 29. Diagrama de la estructura del robot en Simscape (Simulink®), ingresando señal de movimiento.

Se utiliza el bloque S PS que convierte una señal de simulink sin unidades a una magnitud física, para poder ser interpretada por los bloques de simscape, y viceversa para el bloque PS S. Los bloques de tipo junta de revolución cuenta diferentes ajustes como se visualiza en la figura 30.

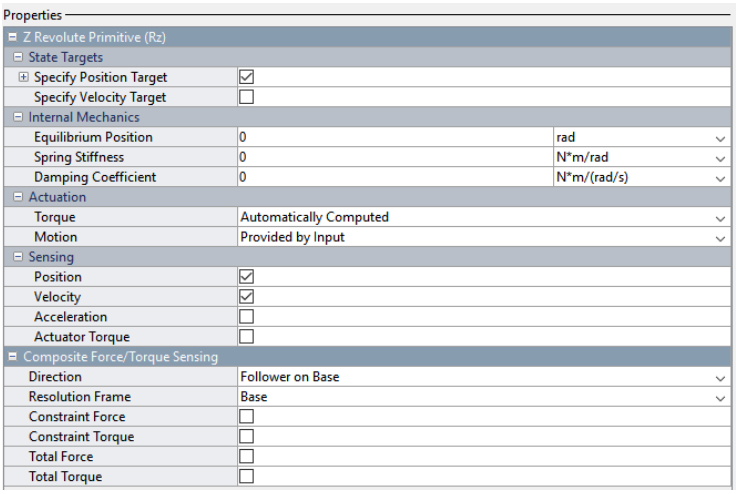


Figure 30. Menú de ajustes del bloque tipo junta de revolución.

Para esta prueba se indica en la sección de actuación, el movimiento proveniente de una entrada y el torque en cálculo automático, en la sección del censado se selecciona la posición y velocidad para poder registrar los valores que se obtengan durante el tiempo de simulación en un osciloscopio. Las gráficas de la izquierda de color rojo de las figuras 31, 32 y 33, muestra las trayectorias generadas con la herramienta de signal builder que se ingresan a las junta hombro, brazo y antebrazo.

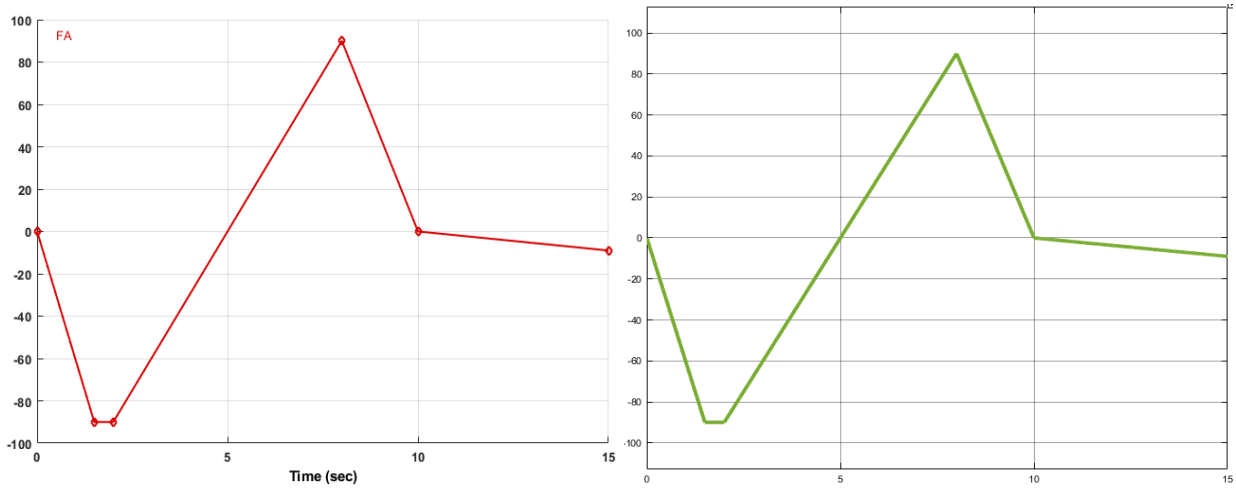


Figure 31. Trayectoria de movimiento ingresada (color rojo), y obtenida (color verde) expresada en grados (DEG), para hombro.

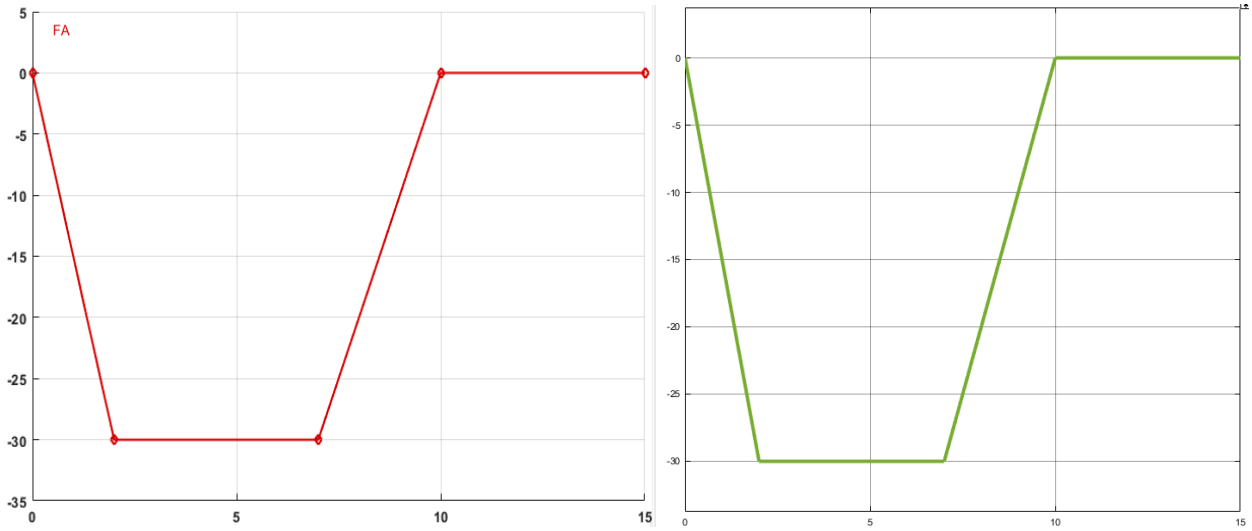


Figure 32. Trayectoria de movimiento ingresada (color rojo), y obtenida (color verde) expresada en grados (DEG), para brazo.

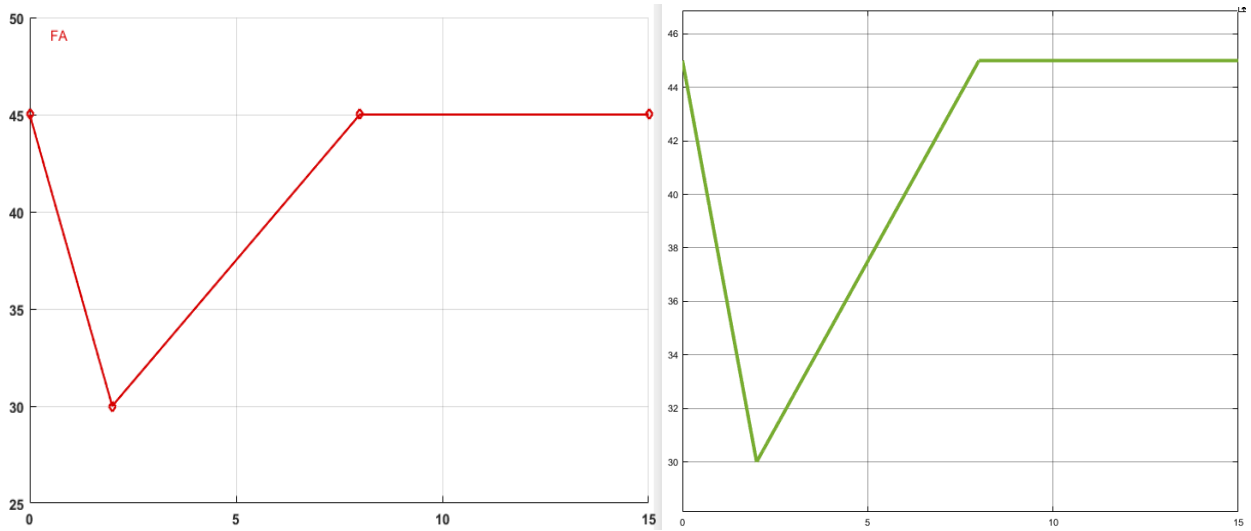


Figure 33. Trayectoria de movimiento ingresada (color rojo), y obtenida (color verde) expresada en grados (DEG), para antebrazo.

Una vez se realiza la simulación del sistema durante un tiempo de 15 segundos, como se observa en la siguientes graficas de color verde de la figura 31, 32 y 33, los movimientos de cada junta siguen la trayectoria ingresada, simulando el movimiento de un motor, como se controla la trayectoria que debe realizar los eslabones, los movimientos no son erráticos como en la prueba anterior sin ningún control. Esto es lo que debe hacer el robot en el entorno virtual una vez modelado los actuadores con su respectivo control, al igual en la realidad con un porcentaje de error muy bajo en el seguimiento de la trayectoria.

2.3 Inconvenientes y recomendaciones

Inicialmente se había diseñado una transmisión planetaria como sistema de trasmisión en la base y hombro, pero se descartó ya que este sistema para que tenga movientes precisos, se tendría que hacer los dientes de los engranes muy pequeños, algo que es poco ideal en materiales como el PLA o aclínico, lo cual se tendría que recurrir al metal, para nuestro caso era un poco complicado de acceder a ese tipo de mecanizado. Por lo cual se optó por un rodamiento que soluciona este inconveniente de una forma más rápida y barata.

En la exportación de la estructura mecánica del robot a simulink con Simscape multiboy, se obtuvo diagrama de bloques bastante extenso, el motivo de esto, es porque durante el ensamble en Solidworks® se ensamblaron solo piezas y no otros ensambles (conjunto de piezas que forman una pieza más compleja), para solucionar este inconveniente se recomienda ensamblar previamente cada eslabón por separado como base, hombro, brazo, antebrazo, pinza, etc y luego ensamblar todo el brazo robótico o el mecanismo que se esté trabajando, esto con el fin de tener un modelo más optimizado ya que un modelo de mayor tamaño requiere más recursos de procesamiento.

En ocasiones el reconocimiento del tipo de juntas en Simscape multibody no es correcto como ocurrió en este caso. Una forma de percatarse es mirar el tipo de juntas en el diagrama si es rotacional o prismática, y reemplazar por la correcta. Otra forma en la que se puede percatar es en la simulación, mediante generación de movimiento de forma automática, el movimiento tiene que afectar todas las juntas que en nuestro caso no afectaba la junta del brazo, el cual se realizó la respectiva corrección en el diagrama.

CAPÍTULO 3, ACTUADORES Y SENSORES

En este inciso se explica cómo fue el proceso de puesta en marcha de los motores, el modelado matemático, simscape y los problemas que se presentaron durante esta etapa.

3.1 Primeros pasos

Inicialmente se procede a leer el manual Motion Controller MBCL 2805 que proporciona el fabricante Faulhaber®, para entender cómo funciona estos servos motores. Una vez terminada la lectura de la guía, se da el primer paso realizando un control de velocidad con el potenciómetro conectado a la entrada análoga del controlador como lo muestra la figura 34, el controlador se encuentra en modo continuo para esta prueba.

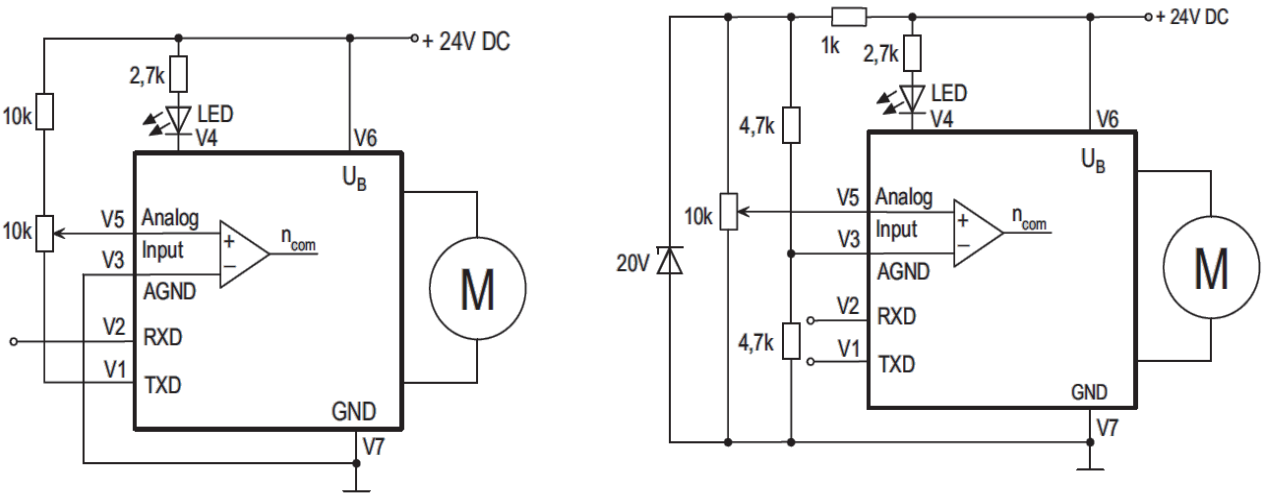


Figure 34. Circuito de control de velocidad mediante potenciómetro.

El diagrama de la izquierda de la figura 34, muestra un circuito más simple, pero cuenta con la desventaja de no tener acceso a la comunicación serial. Según las pruebas realizadas este circuito no tiene tanta precisión. En el diagrama de la derecha, este cuenta con la ventaja que se tiene comunicación serial disponible, y el circuito es más preciso, según las pruebas realizadas. En la siguiente imagen (figura 35) se muestra el montaje de uno de los circuitos.

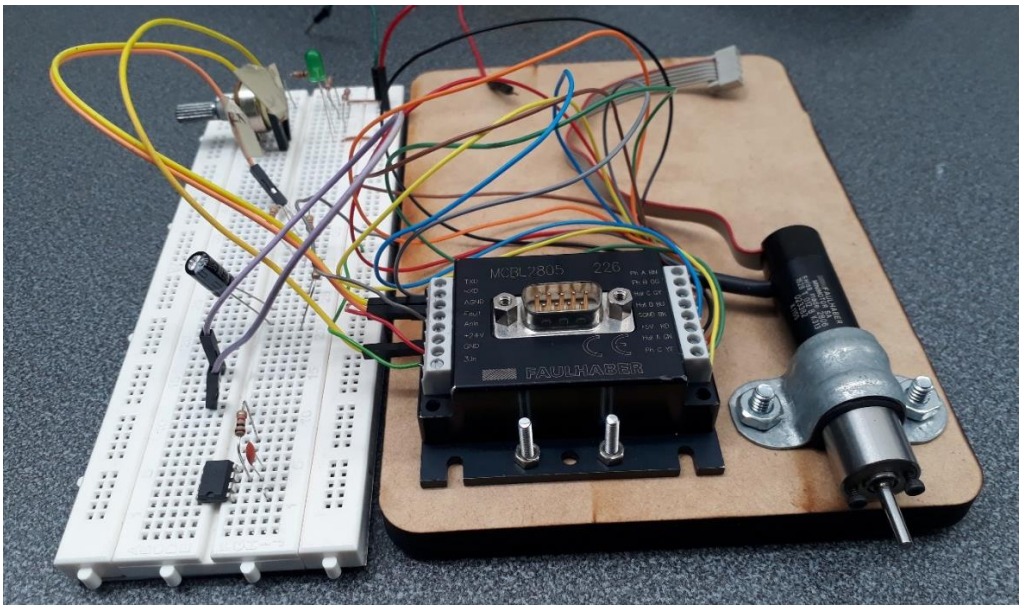


Figure 35. Montaje del diagrama sin acceso a puerto serial, para control de velocidad mediante potenciómetro.

En la figura 36, se muestra la lectura del pin A del encoder externo, esto con el fin de realizar pruebas y dar una mejor interpretación del funcionamiento del mismo.

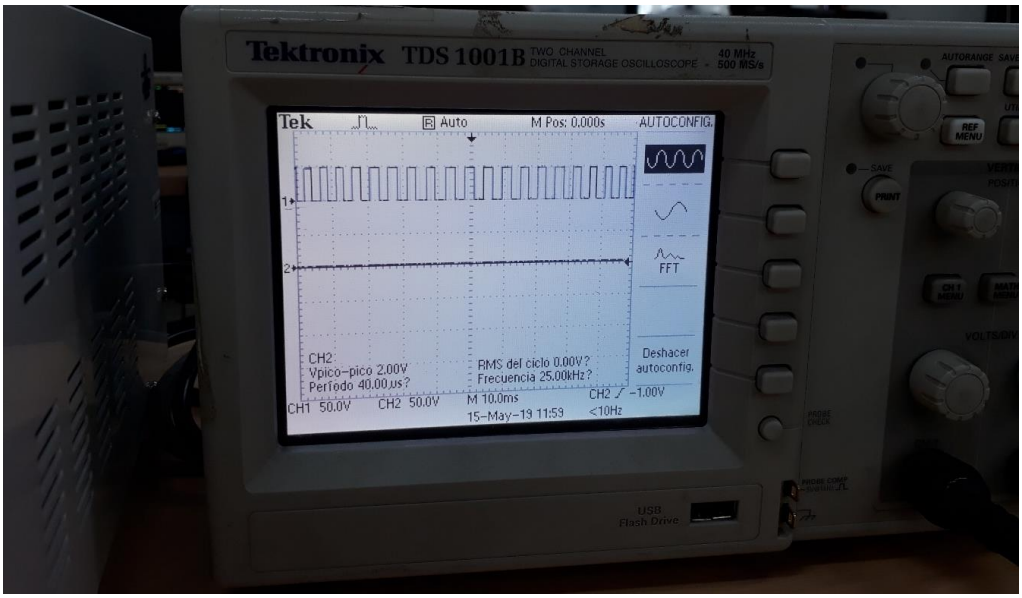


Figure 36. Lectura del puerto A del encoder externo.

Para controlar múltiples motores con motion manager, el fabricante propone la solución de una tarjeta multiplexer que convierte la señal RS-232 a UART que va conectada al pin tx y rx del controlador, es necesario conectar para cada controlador un multiplexer, la figura 3, muestra un esquema de esta tarjeta multiplexer.

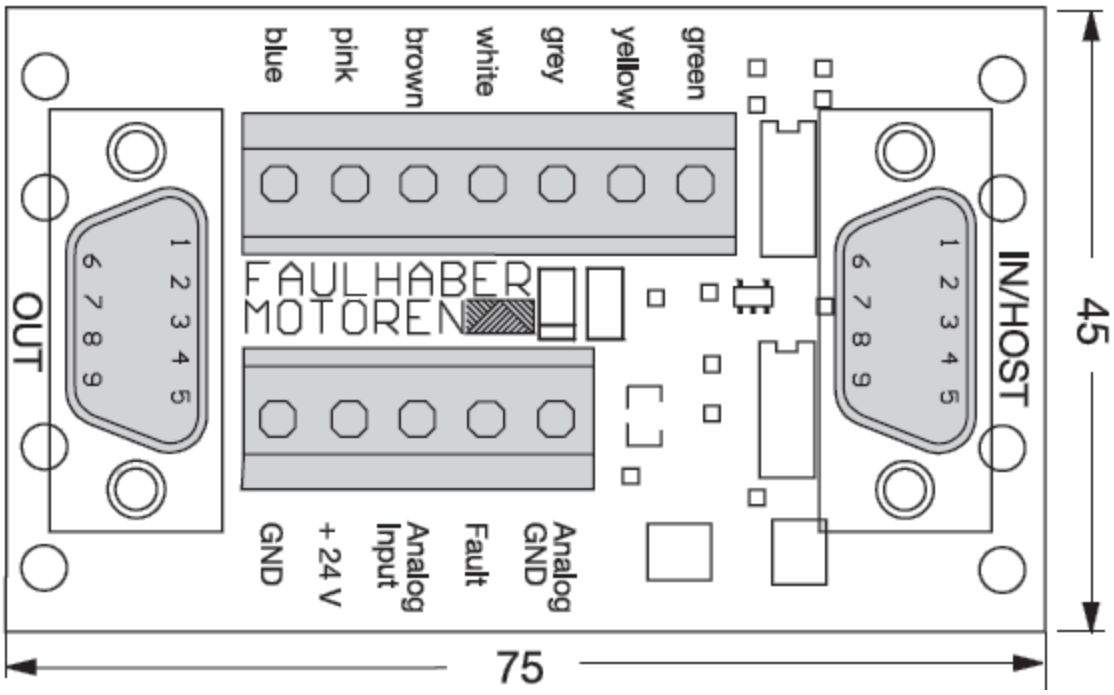


Figure 37. RS-232 Multiplexer Board ***(Faulhaber®, Manual Motion Controller for Brushless DC-Servomotors).***

Para el caso de este proyecto se dispone de 3 motores con sus respectivos controladores, pero solo se dispone de una sola tarjeta multiplexer, el cual se descarta la opción de controlar los motores mediante esta herramienta.

Se exploraron otras opciones como intentar realizar comunicación con el puerto UART mediante Arduino®, enviando por puerto serie comandos ASCII, pero la comunicación no fue exitosa tal vez debido a que esta comunicación trabajaba a valores de tensión diferentes, esto se dice de forma especulativa, pero se midieron los valores de voltaje de la tarjeta multiplexer en el puerto UART y sus valores son de 7.5 V y arduino trabaja con 5V.

Esto nos imposibilita utilizar muchas de las cualidades que tiene el controlador, el cual se tiene recurrir al control mediante la entrada analógica, con acceso al control posición y velocidad externamente, pero no control de par. Imposibilitando aplicar estrategias de control avanzadas como el control por compensación de gravedad.

3.2 Modelo matemático del motor

El modelo del motor sin escobillas como el que se utiliza en el proyecto, no difiere mucho de un modelo de motor de corriente continua convencional, en la figura 39 se representa un circuito equivalente del motor DC convencional.

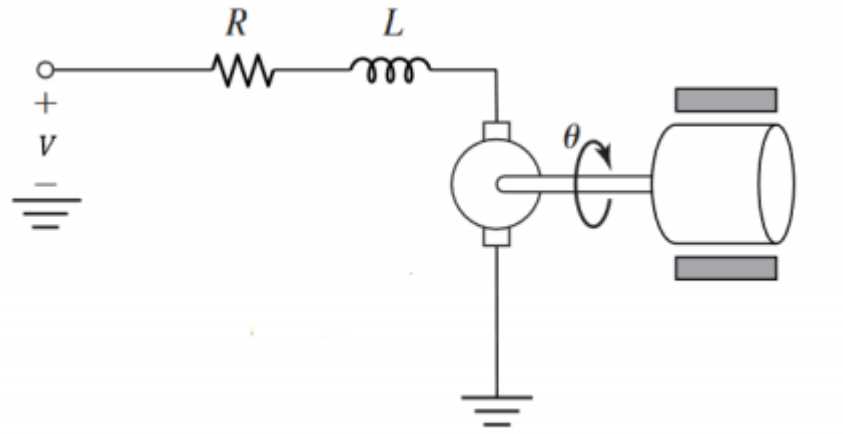


Figure 38. Circuito equivalente motor dc (Duro López & Villarejo Mañas, 2017).

La ecuación que representa el comportamiento eléctrico es.

$$V = Ri + L\frac{di}{dt} + e \qquad \textit{Ec. 2}$$

Donde **V** es tensión de la fuente, i es la corriente de armadura, **e** la fuerza contra electromotriz, R la resistencia y L la inductancia.

La ecuación que representa el comportamiento del modelo mecánico es.

$$T_e = K_f W_m + J\frac{dw_m}{dt} + T_L \qquad \textit{Ec. 3}$$

Donde $\mathbf{W_m}$ es la velocidad angular, $\mathbf{T_e}$ el par eléctrico, $\mathbf{T_L}$ el par de la carga que actúa sobre el rotor, $\mathbf{K_f}$ la constante de fricción del rotor y **J** es la inercia del rotor.

El sistema mecánico para el motor sin escobillas es igual que para el motor DC, entonces a continuación se representará el sistema eléctrico.

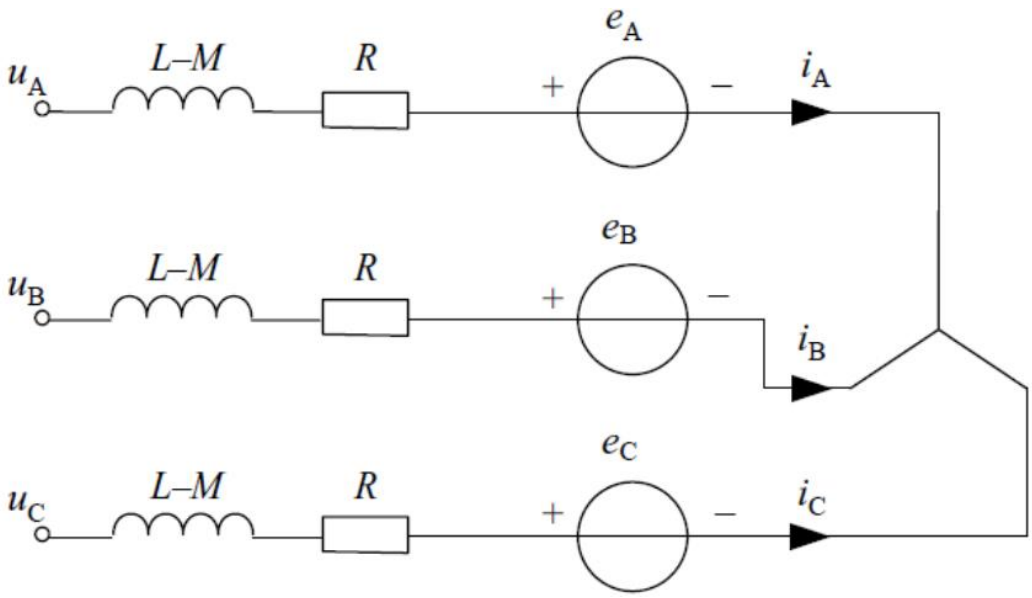


Figure 39. Circuito equivalente motor sin escobillas (Duro López & Villarejo Mañas, 2017).

$$U_A = Ri_A + (L - M)\frac{di_A}{dt} + e_A \qquad \text{Ec. 4}$$

$$U_B = Ri_B + (L - M)\frac{di_B}{dt} + e_B \qquad \text{Ec. 5}$$

$$U_C = Ri_C + (L - M)\frac{di_C}{dt} + e_C \qquad \text{Ec. 6}$$

$$i_A + i_B + i_C = 0 \qquad \text{Ec. 7}$$

Donde R es la resistencia, L la inductancia, y M la inductancia mutua propia de cada fase, que en este caso es la misma para todas las fases, ya que tienen un devanado simétrico. $\boldsymbol{e_A, e_B, e_C}$ son las fuerzas contra electromotriz de cada fase, $\boldsymbol{i_A, i_B, i_C}$ las corrientes de cada fase. Con la ley de tensión de Kirchhoff obtenemos las tensiones de línea $\boldsymbol{U_{AB}, U_{BC}}$ $\boldsymbol{y}$ $\boldsymbol{U_{CA}}$.

$$U_{AB} = R(i_A - i_B) + (L - M)\frac{di_A}{dt} + (M - L)\frac{di_B}{dt} + (e_A - e_B) \qquad \text{Ec. 8}$$

$$U_{BC} = R(i_B - i_C) + (L - M)\frac{di_B}{dt} + (M - L)\frac{di_C}{dt} + (e_B - e_C) \qquad \text{Ec. 9}$$

$$U_{CA} = R(i_C - i_A) + (L - M)\frac{di_C}{dt} + (M - L)\frac{di_A}{dt} + (e_C - e_A) \qquad \text{Ec. 10}$$

La fuerza contra-electromotriz inducida en los devanados del estator y el par eléctrico, se expresan de la siguiente manera.

$$e_A = \frac{K_e}{2}F(\theta) \qquad \text{Ec. 11}$$

$$e_B = \frac{K_e}{2}F\left(\theta - \frac{2\pi}{3}\right) \qquad \text{Ec. 12}$$

$$e_C = \frac{K_e}{2} F\left(\theta - \frac{4\pi}{3}\right) \qquad \textit{Ec. 13}$$

$$T_e = \frac{K_e}{2}\left(F(\theta) i_A + F\left(\theta - \frac{2\pi}{3}\right) i_B + F\left(\theta - \frac{4\pi}{3}\right)\right. \qquad \textit{Ec. 14}$$

Donde $\boldsymbol{K_e}$ es la constante de la fuerza contra electromotriz, que se representa de forma trapezoidal a lo largo del ángulo eléctrico, que viene definida por la función $\boldsymbol{F(\theta)}$ que está acotada en el intervalo [-1,1], el periodo de esta función está definida de la siguiente manera.

$$F(\theta) \begin{cases} 1 & 0 \le \theta < \frac{2\pi}{3} \\ 1 - \frac{6}{\pi}\left(\theta - \frac{2\pi}{3}\right) & \frac{2\pi}{3} \le \theta < \pi \\ -1 & \pi \le \theta < \frac{5\pi}{3} \\ -1 + \frac{6}{\pi}\left(\theta - \frac{5\pi}{3}\right) & \frac{5\pi}{3} \le \theta < 2\pi \end{cases} \qquad \textit{Ec. 15}$$

El ángulo eléctrico $\boldsymbol{\theta}$ del campo inducido en el estator, está relacionado directamente con el ángulo mecánico $\boldsymbol{\theta_m}$ determinado sobre la circunferencia del motor, al igual la velocidad del rotor y su posición estarán relacionadas de igual manera, y donde $\boldsymbol{p}$ es el número de polos del motor, como se muestra en las siguientes ecuaciones (Duro López & Villarejo Mañas, 2017).

$$\theta = \theta_m \frac{p}{2} \qquad \textit{Ec. 16}$$

$$\frac{d\theta}{dt} = \frac{p}{2} W_n \qquad \textit{Ec. 17}$$

La potencia electromagnética $\boldsymbol{P_e}$ es la potencia de transferencia al rotor, y es igual a la suma productos de corriente y la fuerza contra electromotriz de las fases como se indica en la ecuación 19. (Duro López & Villarejo Mañas, 2017)

$$P_e = T_e W_m \qquad \textit{Ec. 18}$$

$$T_e = \frac{i_A e_A + i_B e_B + i_C e_C}{W_m} \qquad \textit{Ec. 19}$$

Recordando que W_m es la velocidad de giro**.**

3.3 Modelo de los actuadores en Simscape Electronics

Mediante la herramienta Simscape Electronics se propone un modelo de los actuadores (figura 41) basado en el seminario web de MathWorks® (Osorio, How a Differential Equation Becomes a

Robot, Part 2: Actuators and Sensors, 2012), permitiendo modelar de forma rápida un motor y respectivo amplificador con los bloques que dispone esta herramienta.

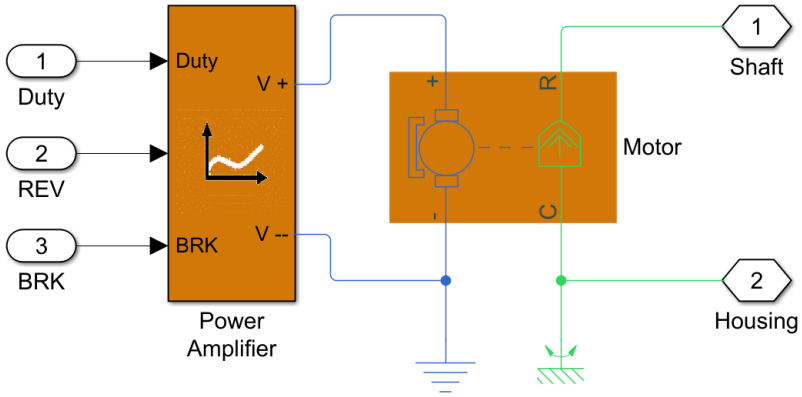


Figure 40. Modelo del motor con su amplificador en Simscape electronics.

El bloque del motor permite ingresar parámetros de ajustes como la resistencia de armadura, inductancia, par de torsión, inercia del rotor, amortiguación del rotor. La figura 42, representa el amplificador del motor.

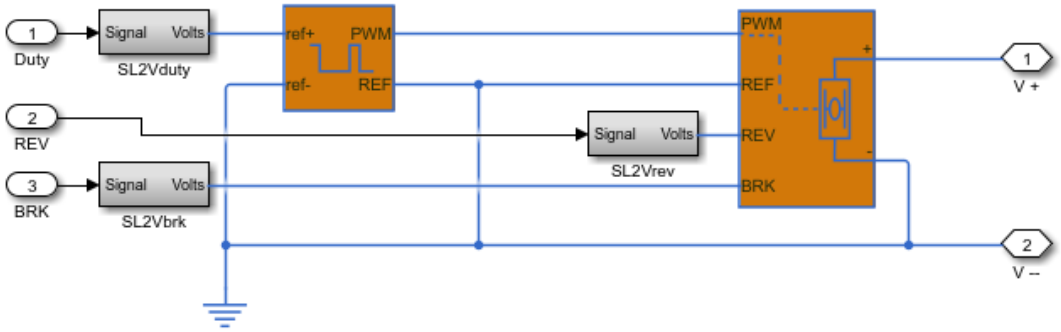


Figure 41. Modelo del motor con su amplificador en Simscape electronics.

El modelo del amplificador dispone de un driver, se pueden configurar parámetros como voltaje de umbral, amplitud de señal PWM, Voltaje de umbral de inversión, Tensión de umbral de frenado, amplitud de voltaje de salida, puente total sobre la resistencia, diodo de marcha libre sobre la resistencia. También dispone de un controlador pwm que nos permite ingresar parámetros como la frecuencia del PWM, voltaje de entrada del ciclo de trabajo, Amplitud de voltaje de salida. La señal de entrada simulink es convertida en una magnitud física, específicamente en valores de voltaje para poder ser interpretado por los bloques de simscape, como se muestra en la figura 43.

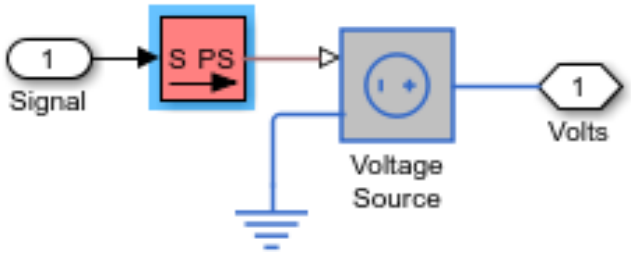


Figure 42. Conversión de señal de Simulink a valores de voltage.

La salida R y C del modelo del motor figura 41, son puertos que representa la rotación mecánica del motor, el bloque de sensor de par ideal de la figura 44, convierte una variable que pasa a través del sensor en una señal de control proporcional al par, con un coeficiente de proporcionalidad especificado. El sensor es ideal ya que no tiene en cuenta la inercia, la fricción, los retardos, ni el consumo de energía, etc.

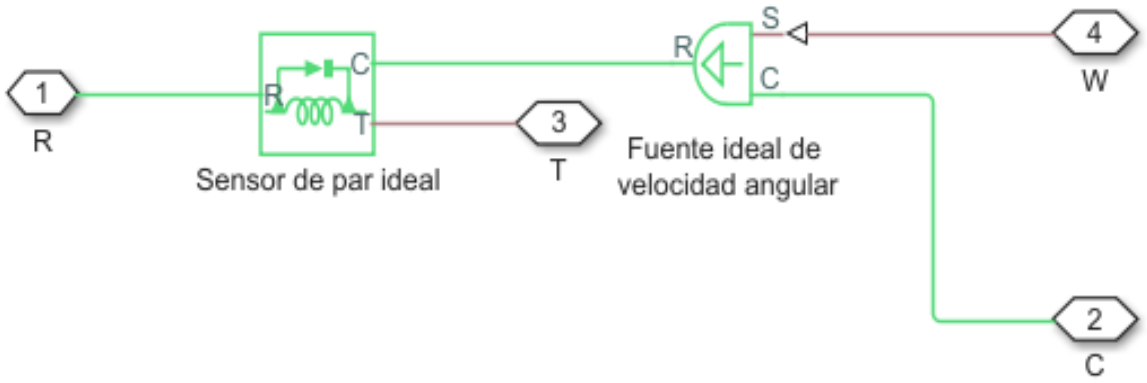


Figure 43. interfaz que convierte la señal mecánica a una señal proporcional a torque.

El bloque de fuente ideal de velocidad angular, genera un diferencial de velocidad en sus terminales proporcional a la señal de entrada física. La fuente es ideal en el sentido de que se supone que es lo suficientemente potente como para mantener la velocidad especificada, independientemente del par que se ejerza en el sistema. En la figura 44, se muestra la implementación de los actuadores con la estructura del robot, usando subsistemas para tener un diagrama más ordenado.

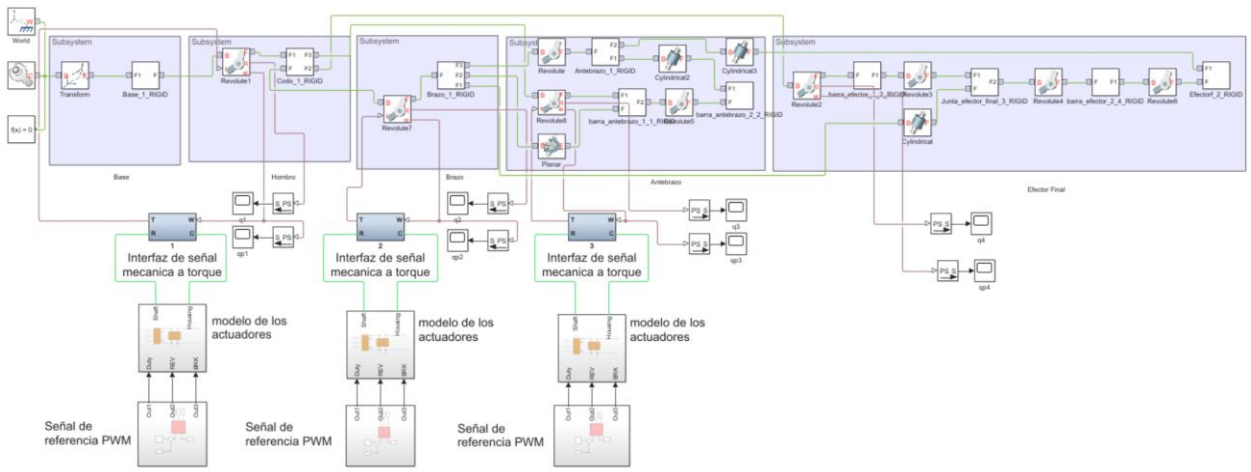


Figure 44. Diagrama estructura del robot con actuadores en Simscape (Simulink®).

A los actuadores se les ingresa una tensión de excitación mediante un generador de pulsos a cada actuador con su respectiva referencia (subsistema señal de referencia PWM), como se muestra en la siguiente imagen, estos valores son aleatorios para esta prueba.

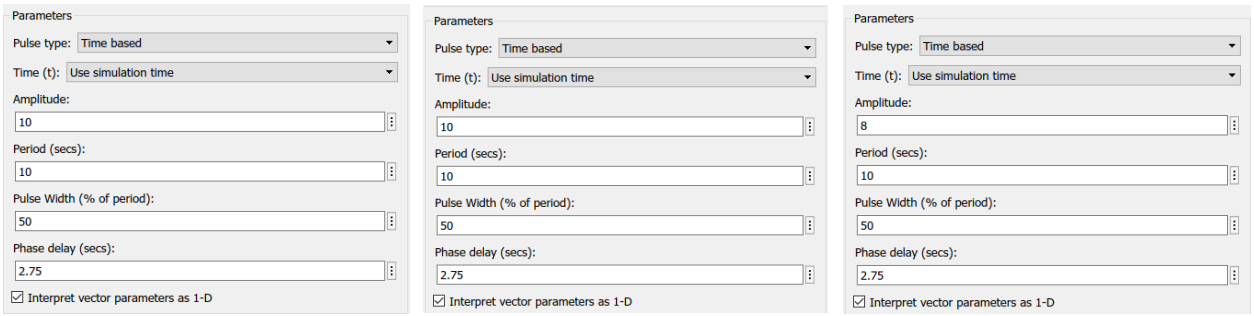


Figure 45. Valor de los generadores de pulsos hombro, brazo y antebrazo (Simulink®).

La figura 46, representa la posición de cada eslabón en un tiempo de simulación de 10 segundos. Donde la gráfica de posición hombro de forma lineal, va desde un tiempo 0 a 2.8 segundos a una posición -1.7 radianes, luego de un tiempo de 7.8 segundos a una posición de 4.1 rad, y para finalizar termina en 10 segundos con una posición 2.2 rad. La grafica de posición del hombro desde el tiempo 0 inicializa en una posición de 0.21 radianes y un tiempo de 4.1 segundos tiene un pico máximo de 0.68 radianes, bajando a 0.35 rad en un tiempo 0.68 segundos, y luego moviéndose a una posición de 0.51 radianes en un tiempo de 8.8 segundos y terminando en un valor aproximado de 0.43 rad en el tiempo de 10 segundos. Para terminar la gráfica de posición del antebrazo

inicializa en 1.2 radianes y se dirige a -0.2 rad en el tiempo 2.8 segundos, alcanza un valor 2.2 radianes en el tiempo 4.8 segundos y se mantiene hasta 7.8 segundos luego va aproximadamente 2.2 radianes en el tiempo 10 segundos.

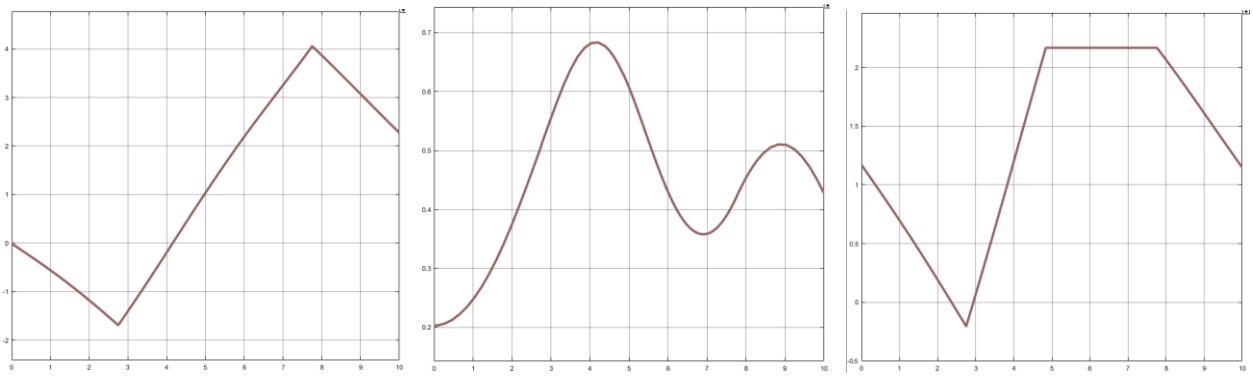


Figure 46. Graficas de posición, de hombro, brazo y antebrazo, simulación de 10 segundos, unidades en rad (Simulink®).

La figura 47, representa las gráficas de velocidad para hombro, brazo y antebrazo, representando los valores positivos y negativo la dirección de giro del motor en radianes por segundo.

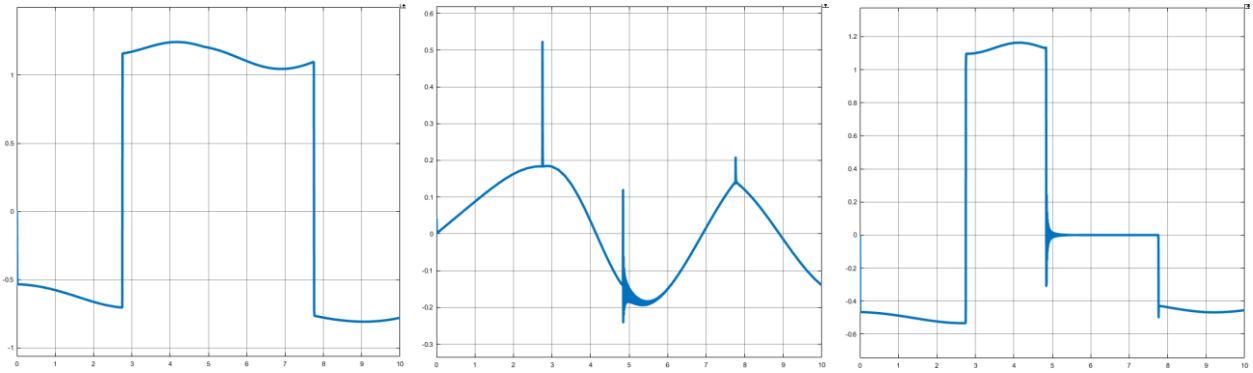


Figure 47. Graficas de velocidad, de hombro, brazo y antebrazo, simulación de 10 segundos, unidades en rad (Simulink®).

Los valores de referencia ingresadas a los actuadores como se menciona anteriormente es de forma aleatoria, con el objetivo de ver el funcionamiento de los mismo como forma de prueba.

3.4 Inconvenientes y recomendaciones

Uno de los mayores inconveniente que se presentó al inicio es respecto a los controladores de los motores, ya que se tuvo que renunciar a su programación nativa debido a que solo se dispone

de una sola tarjeta multiplexer RS-232, y se requiere una para cada motor, sacrificando muchas de las bondades que ofrece el controlador, por lo cual se recurre a diseñar un control externo de posición y velocidad mediante Arduino® y aprovechando el amplificador del controlador de movimiento MCBL 2805 de Faulhaber®.

El otro gran inconveniente que se tuvo, fue la avería de uno de los motores durante el proceso de calibración que ofrece la herramienta Motion Manager de Faulhaber®.

Figure 48. Graficas de posición y velocidad de hombro, brazo y antebrazo en un tiempo de 10 segundos.

Específicamente la avería ocurrió con unos de los motores de la referencia 2206 (figura 48), el daño causado es debido la sintonización de los parámetros del motor y controlador, ya que al girar a altas revoluciones en un corto plazo de tiempo, lo que ocurre es que se traban algunos de los engranes que conecta la caja reductora con el motor, tal vez sea debido a la posición en la que estaba inicialmente el motor, y algún golpe ocasionado anteriormente y otra cosa a mencionar es que el fabricante recomienda no realizar esta sincronización con motores con caja reductora planetaria para evitar estos inconvenientes.

Para solucionar esto se procede a desarmar el motor con sumo cuidado, ya que se maneja engranes y tornillos muy pequeños de difícil acceso, además que se cuenta con una reducción 157:1 lo cual esta caja consta de una transmisión planetaria de varios niveles. En la siguiente imagen (figura 49) se muestra uno de los niveles de la transmisión.

Figure 49. Engranes de la transmisión planetaria de la caja reductora.

La parte donde causo más problemas fue quitar la sección donde se une la caja de trasmisión y el motor (figura 50), debido a la posición de los motores ya que el cuerpo metálico cubre una tercera parte de los tornillos.

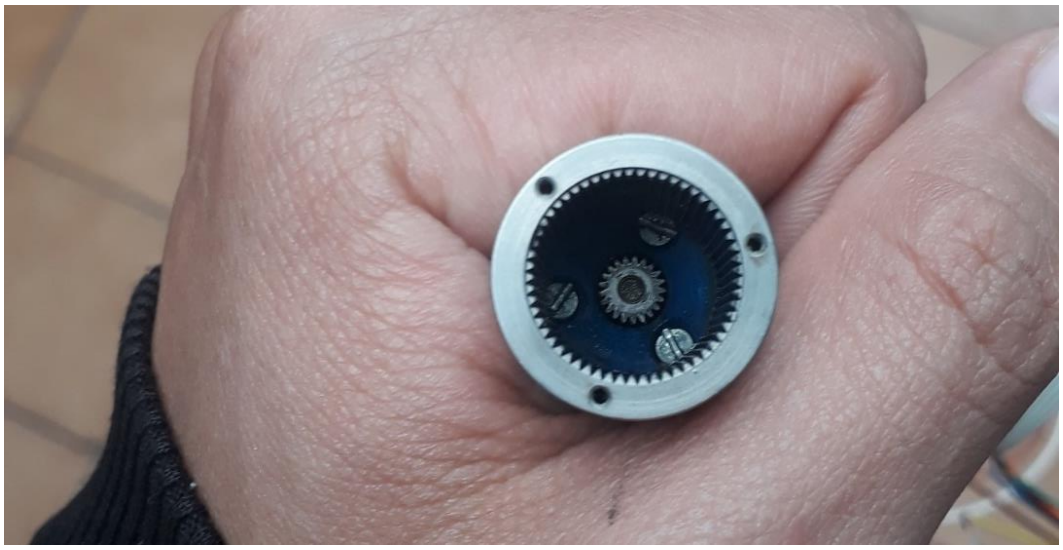


Figure 50. Tornillos que sujetan la caja reductora y el motor brushless.

La recomendación para quitar la sección de la caja reductora es dar una vuelta cada tornillo, y así mantener la presión más o menos igual en toda la circunferencia, evitando la presión en unos de sus lados, una vez acomodado los engranes que se salerón de su lugar, se procede a la lubricación y armado de forma cuidadosa debido a los pequeños tamaños de las piezas, y para finalizar se pone en marcha de nuevo.

CAPITULO 4, CONTROL DE LOS ACTUADORES

Como se menciona anteriormente se tiene como opción el control externamente, el cual se propone diseñar con controlador de posición y velocidad como lo indica el diagrama de bloques de la figura 51. Para llevar esto acabo es necesario que el controlador este en modo control de velocidad y no control de posición, ya que si se controla en este modo no tendremos acceso al control de velocidad. De manera que si está en modo velocidad se puede realizar control de posición mediante un microcontrolador o tarjeta embebida como Arduino®.

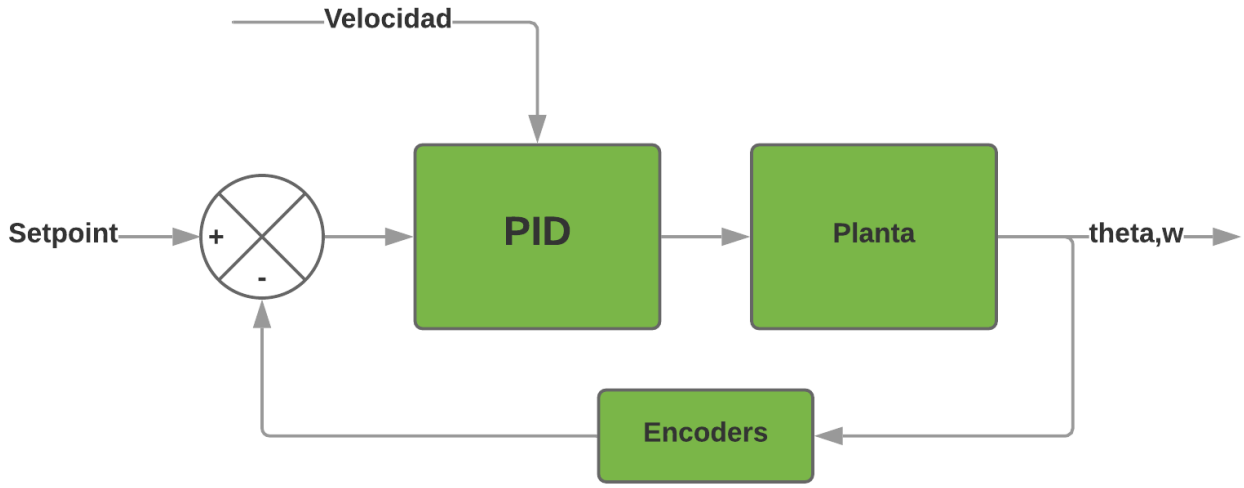


Figure 51. Diagrama de bloques de control de posición y velocidad.

El control de velocidad se realiza mediante feed-forward, ingresando una referencia de velocidad que se obtiene de forma manual, o mediante los sistemas de control cinemático. Se utiliza el control PID ya que permite un mejor control en el tiempo de respuesta, aumenta la presión de la respuesta, y permite ajustar las variables con facilidad, ya que con la contante derivativa es posible de predecir las otras variables, y además realiza un ajuste de forma automática de los valores de los valores integrales que se excedan, ventajas bastantes útiles donde se requiera sintonización de forma rápida para la realizar pruebas.

4.1 control de Posición y velocidad en arduino

El controlador de posición PID propuesto para este proyecto está basado en el algoritmo publicado por Brett Beauregard (Beauregard:, 2015). El filtro para el controlador es basado del sitio de Proyectos de Robóticos (roboticos, 2015). Este filtro es bastante útil para suavizar la

respuesta del motor al llegar al punto designado. Este algoritmo se implementa en la plataforma Arduino® y se utiliza la tarjeta Mega 2560 Rev3 ya que esta cuenta con la característica de tener tres timers con 6 pines para interrupciones que son los pines 2 y 3, 18 y 19, 20 y 21. Esta tarjeta resulta ideal para este proyecto ya que se utilizan tres encoders externos incrementales efecto hall. Además, que esta plataforma cuenta con soporte en otras plataformas como Matlab que se utiliza en este proyecto.

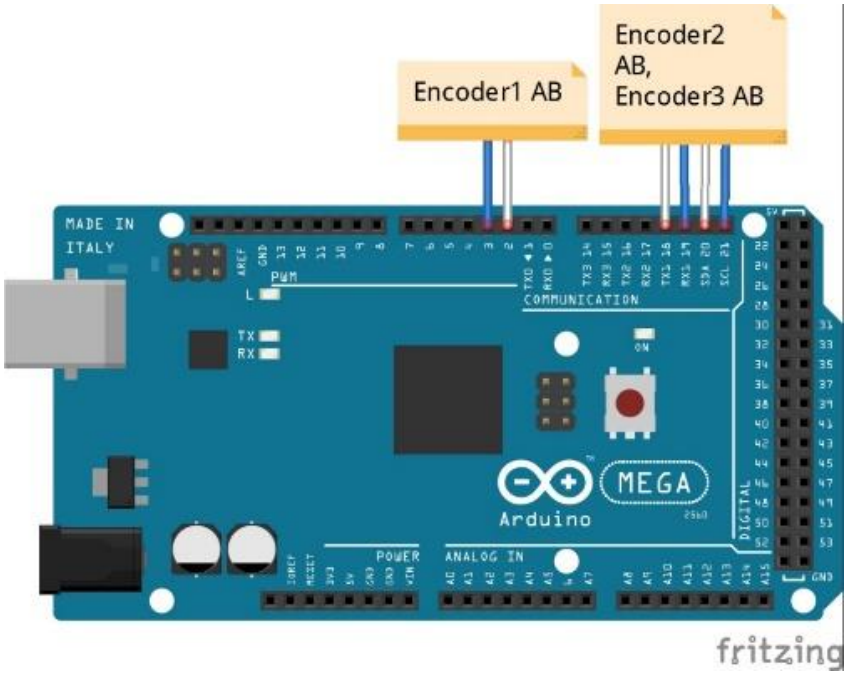


Figure 52. Pines de interrupciones Arduino Mega 2560, conectados a encoder.

A continuación, se explica las partes importantes del funcionamiento código mediante un diagrama de flujo (figura 53) implementado en Arduino® para el control de posición y velocidad. Para empezar, declaramos las entradas y salidas, los valores de las contestes PID, y otros variables extras que se utilizan, una vez declaradas estas variables se inicializa los puertos y se configura la velocidad de conexión serial en baudios, también se declaran las interrumpes que se utilizan para contar las revoluciones e interpretar el sentido de giro mediante los encoder externos que dispone los actuadores, seguido un loop que cuenta con ajustes de control de sentido de giro del motor, ingreso de la referencia y el llamado de funciones como las del cálculo PID y sintonización.

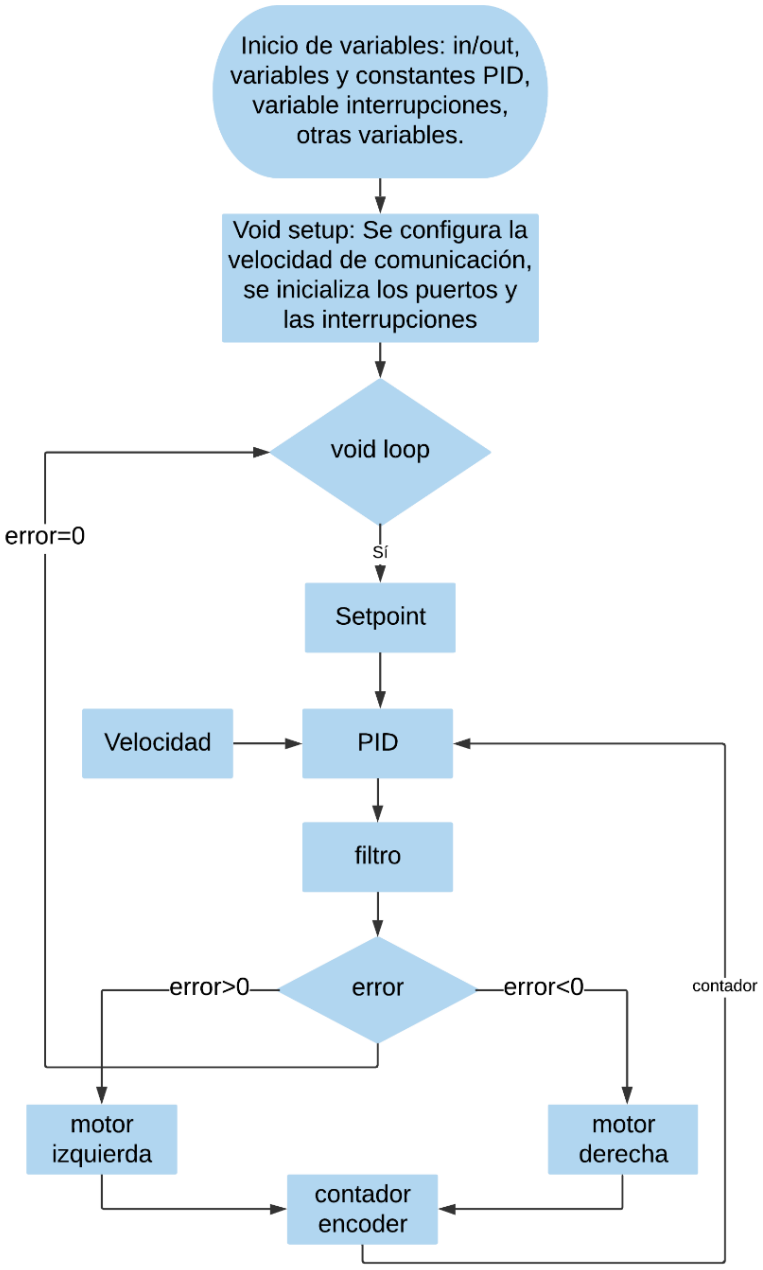


Figure 53. Diagrama de flujo del control de posición y velocidad implementado en arduino.

Para una explicación más detallada el anexo A, contiene el código implementado en arduino.

4.2 Pruebas en simulink

Se implementa un diagrama de bloques básico de prueba para control de posición de un solo motor para realizar pruebas, pero se utiliza la antigua librería de Legacy MATLAB and Simulink Support for Arduino de Giampiero Campa, esta librería no tiene soporte para las nuevas versiones

de Matlab, por lo cual se migra a una versión anterior para poderla utilizar, específicamente la versión 2015ª. La razón de utilizar esta librería es que la versiones Simulink Support Package for Arduino Hardware v18.1.1 de MathWorks Simulink Team, y MATLAB Support Package for Arduino Hardware 18.1.2 de MathWorks MATLAB Hardware Team y versiones anteriores solo permiten realizar la lectura de no más de dos encoders en arduino Mega. (Mathworks®, Arduino Support from Simulink) (Mathworks®, Arduino Support from MATLAB)

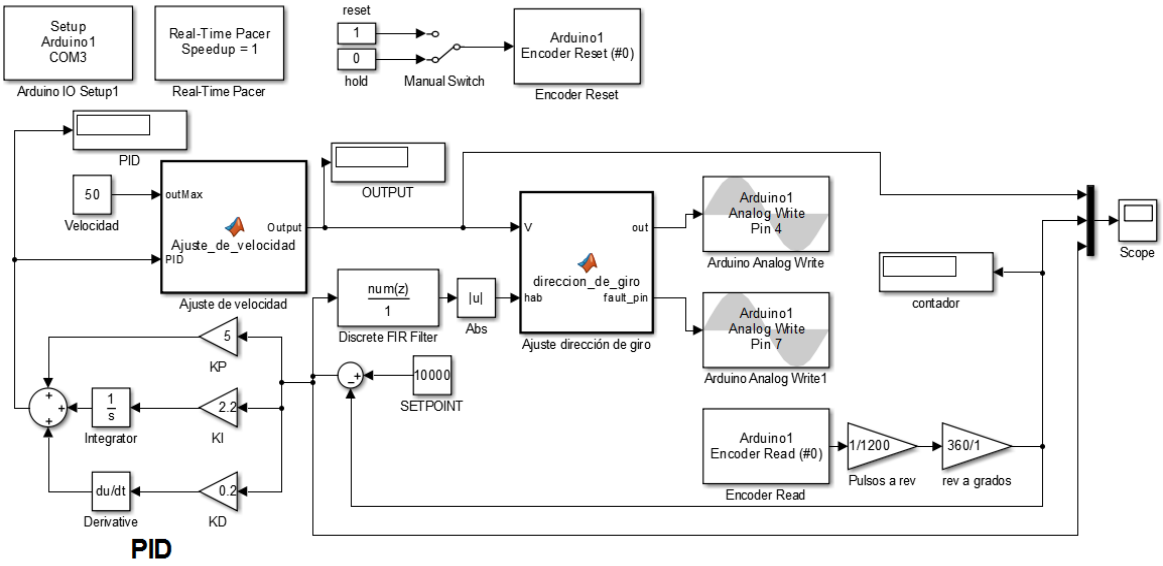


Figure 54. Diagrama de bloques control de posición en Simulink®.

En el primer bloque, la función establece la velocidad máxima en un valor 0 a 255, en un principio para realizar pruebas se deja el valor fijo, pero se puede ingresar deferentes valores para controlar la velocidad del motor, según la aplicación por ejemplo donde se requiera hacer control de movimiento coordinado de múltiples motores, a continuación, se muestra el contenido del bloque ajuste de velocidad.

```
function Output    = Ajuste_de_velocidad(outMax,PID)
outMin = -outMax;
Output = PID;
if (PID > outMax)
    Output = outMax;
else if (PID < outMin)
        Output = outMin; % Acota la salida para que el PWM pueda estar entre
outMin y outMax.
    end
end
end
```

El segundo bloque la función permite controlar el sentido de giro del motor o dejar de mandar una señal si el error es cero.

```
function [out,fault_pin]  = direccion_de_giro(V,hab)
th = 2;
out = 0;
if (hab >= th)
    if (V >= 0)
        fault_pin = 1;  % Cambio de giro
        out = abs(V);
    else
        fault_pin = 0;
        out = abs(V);
    end
else
    out = 0;
    fault_pin = 0;
end
end
```

Al realizar pruebas en tiempo real por medio de comunicación serial a 115200 baudios, sin general y cargar el programa en el arduino, la respuesta de funcionamiento no es lo suficiente optima en valores de velocidad medios o superiores ya que se demora en el alcanzar la referencia debido a la oscilación prolongada hasta que el error sea cero como se muestran en las siguientes figuras 55. Con este tipo de comunicación depende de la disponibilidad de la CPU ya que si se está realizando otras tareas se demora más en alcanzar la referencia.

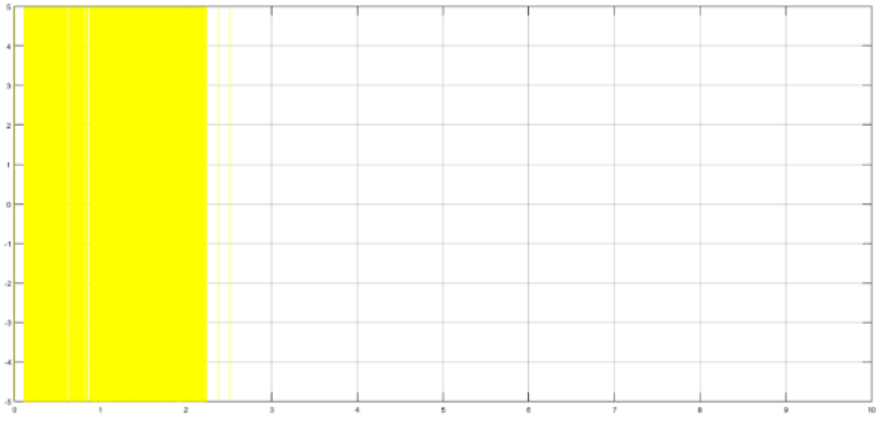


Figure 55. Señal de salida de -5 a 5 voltios.

Esto es debido a las siguientes condiciones; la comunicación tanto en lecturas y escrituras a la Arduino Mega en frecuencias altas afecta a la eficiencia debido a que son motores brushless, se

atenúa este problema cuando son velocidades bajas, pero para la aplicación de este proyecto no es considerable. El siguiente inconveniente es la capacidad de cálculo del procesador que en tareas más pesadas como trayectorias que se realizan en este proyecto este modo de operación es ineficiente. Por lo cual es imprescindible cargar el programa en el arduino.

A la hora de realizar pruebas con el programa corriendo en el arduino, se presenta un inconveniente más, se presentan muchos problemas a la hora de generar y cargar el código al arduino. Por lo que se optó como solución utilizar el código del IDE de arduino.

4.3 Control del sistema (Simulación)

En este apartado se propone un control de posición para controlar el sistema en un entorno de simulación simulink®, en la figura 56 se visualiza los subsistemas que conforma la planta y el controlador.

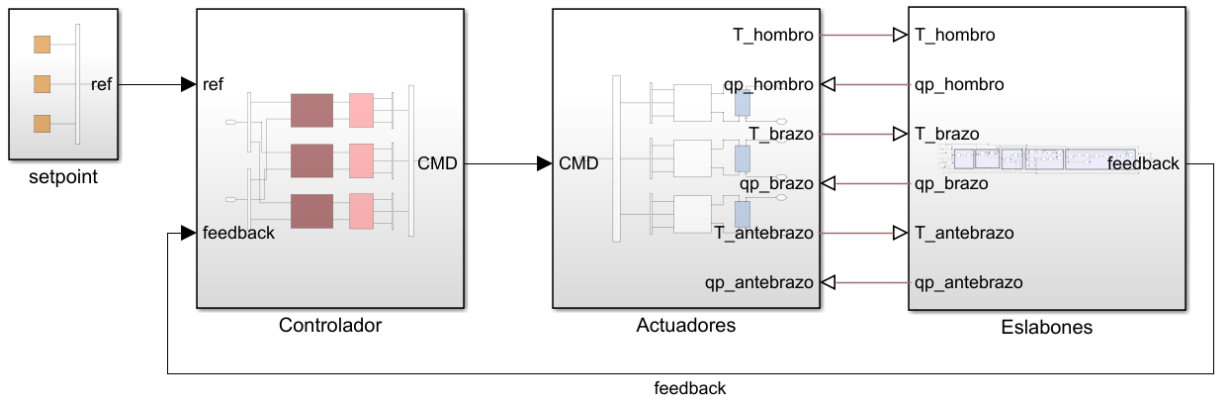


Figure 56. Diagrama del sistema con control básico en Simulink®.

La figura 57, se muestra los subsistemas PID y los subsistemas de voltaje de excitación de los actuadores. Este tipo de controlador es basado en el seminario web de How a Differential Equation Becomes de MathWorks® (Osorio, 2012), el principio de funcionamiento no difiere mucho del propuesto anteriormente a diferencia del filtro antialiasing (figura 58) que se utiliza.

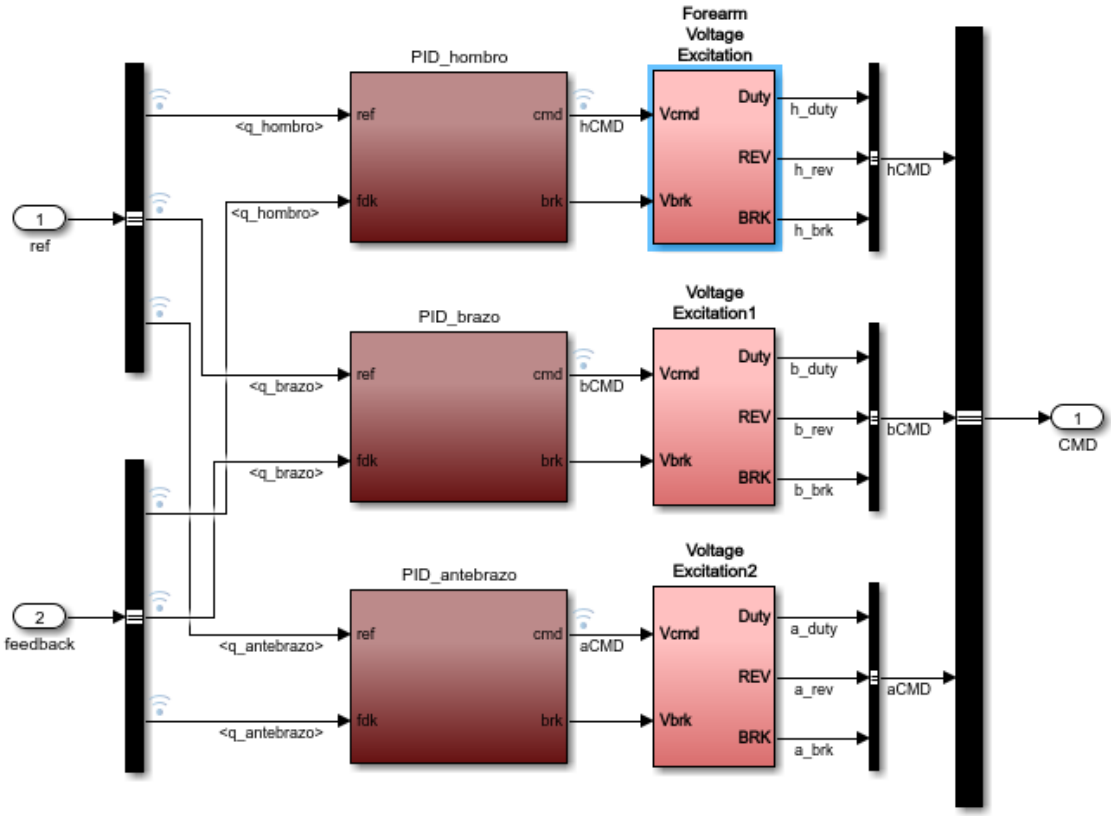


Figure 57. Controlador, Simulink®.

El subsistema del control PID está conformado por un filtro y un bloque PID el cual este permite realizar auto ajuste al sistema u colocar valores de las contante manualmente. El filtro antialiasing basado en la sección Actuators and Sensors del seminario web de How a Differential Equation Becomes (Osorio, 2012). atenúa las frecuencias altas, permitiendo suavizar la respuesta de la señal. Básicamente es un filtro análogo (tiempo continuo).

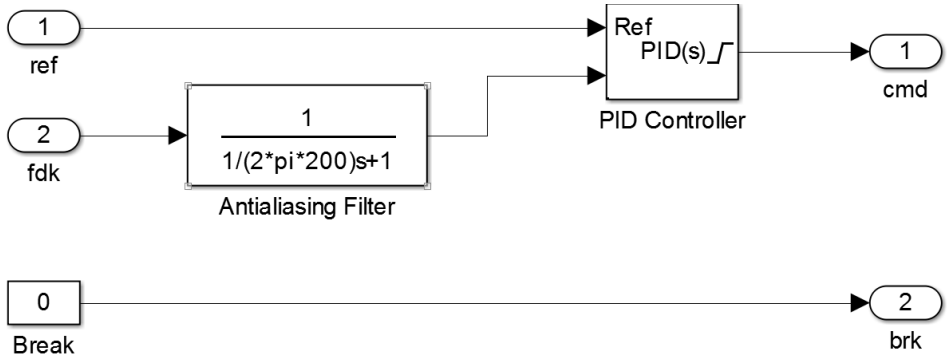


Figure 58. Control PID más filtro basado en (Osorio, 2012).

El subsistema de voltaje de excitación (figura 59), lo que hace es mandar una señal de 5V o 0V por el puerto Duty dependiendo del error, si el error es cero su valor de salida es OV si no lo es independiente si es negativo o positivo ya que aplica un bloque de valor absoluto manda una señal 5v esto mediante el bloque de saturación, en principio este valor de tensión puede ser variable para el control de la velocidad, pero en esta prueba se trabaja con valores fijos. El puerto REV nos invierte el sentido de giro mandando una señal de 5V mediante una condición, en este caso si el error está en los valores negativos. Si el voltaje del puerto BRK es mayor que el voltaje de umbral de frenado, entonces los terminales de salida se cortocircuitan.

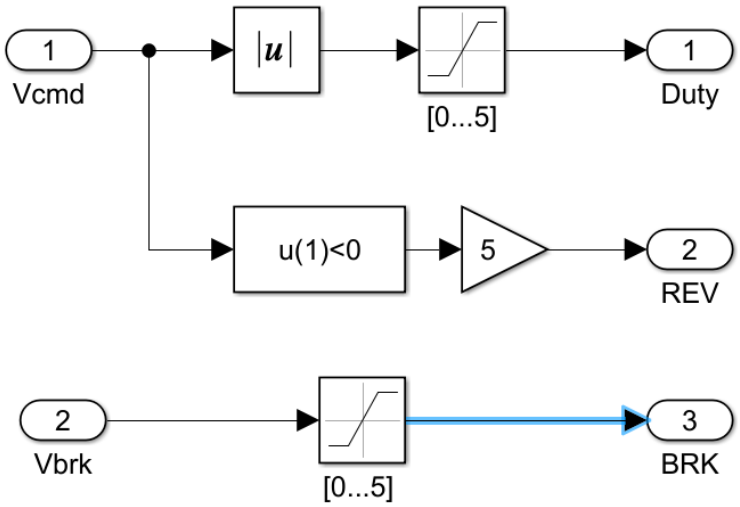


Figure 59. Generación señal de excitación e inversión de giro del motor Simulink®.

Como vemos el principio de funcionamiento es muy similar al circuito básico de control de posición realizo primeramente en simulink® de la figura 54, si no la diferencia radica que estas condiciones se establecieron mediante código. En la figura 60, se muestra los bloques que conforma el subsistema de señal de referencia.

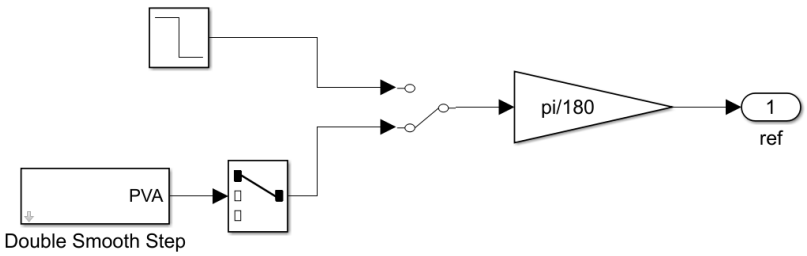


Figure 60. Señal de referencia en rad, Simulink®.

4.3.1 Sintonización PID.

Para la sintonización de cada uno de los controladores PID, se realiza con la herramienta que ofrece simulink PID tunner.

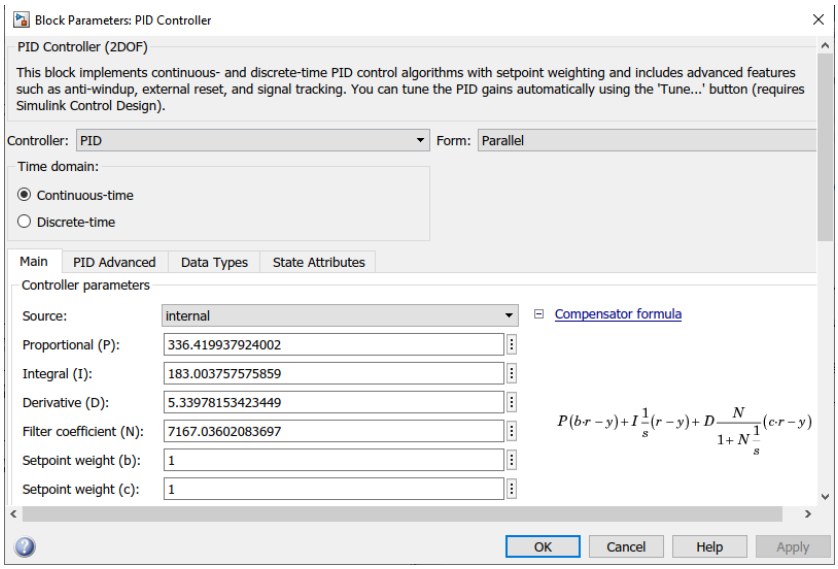


Figure 61. PID controller, Simulink®.

Esta herramienta nos permite ajustar los parámetros de una forma manual con la opción de PID Tuner, ya que nos permite visualizar el efecto que tiene el controlador como se muestra en la siguiente figura.

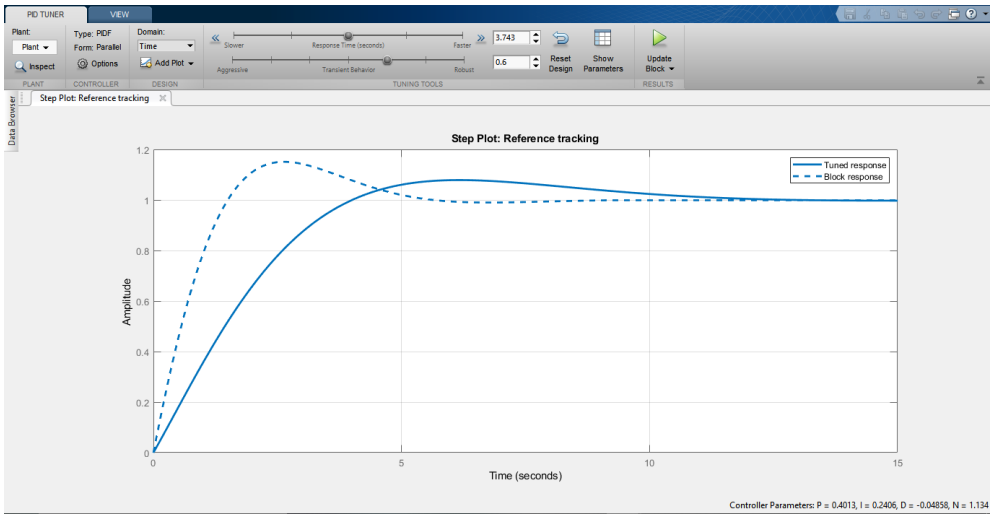


Figure 62. PID Tuner, Simulink®.

Valores PID; anexo k.

También cuenta con la opción de auto tunner que para algunas aplicaciones puede resultar bastante conveniente, pero la forma manual es bastante rápida y eficiente a comparación de otros métodos de sintonización, como ziegler nichols, o asignación de polos.

4.3.2 resultados de la Simulación.

Las gráficas de las figuras 63,64 y 65 muestran la señal de referencia ingresadas al sistema y las señales de salidas obtenidas, las gráficas terminadas en fdk son las obtenidas en la salida.

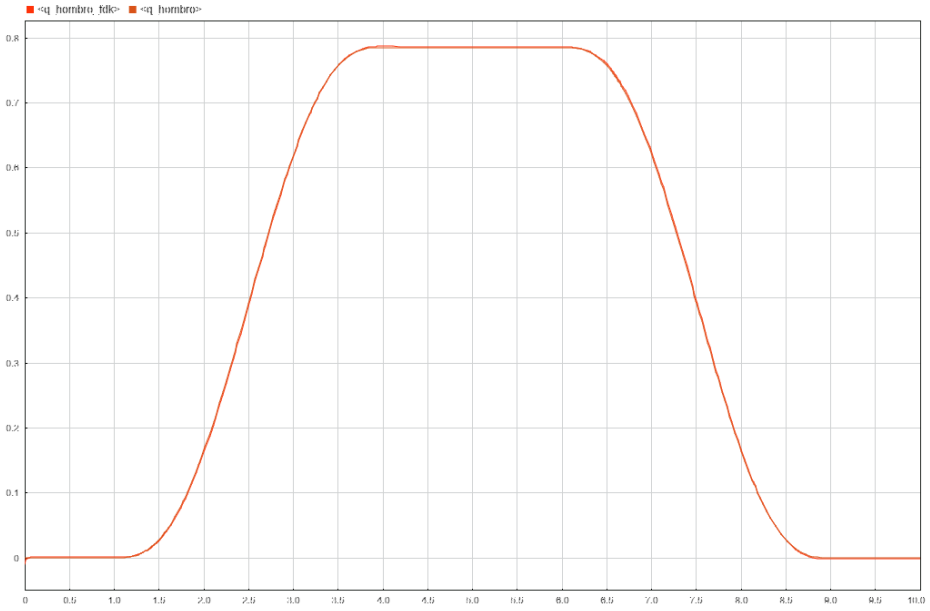


Figure 63. Señal de entrada y salida del hombro (Simulink®).

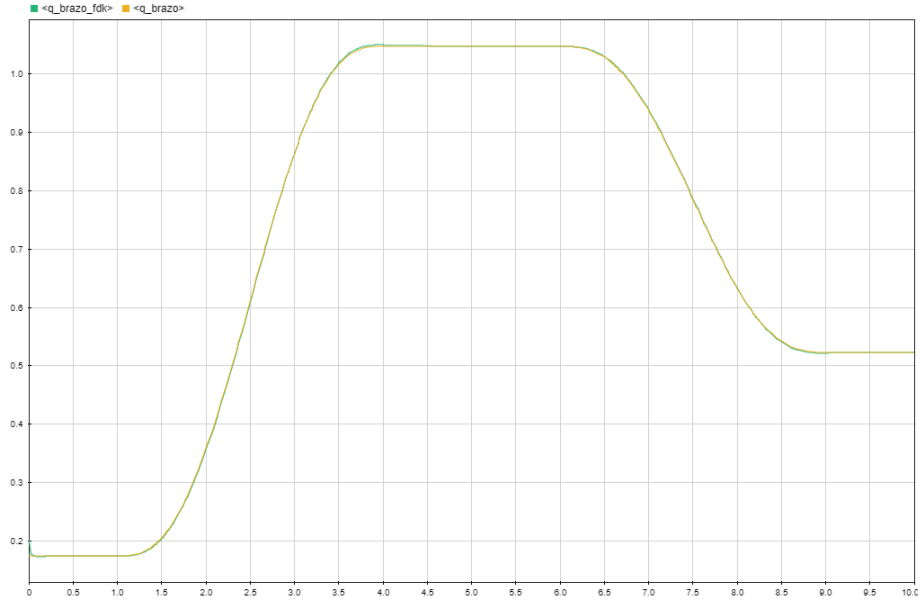


Figure 64. Señal de entrada y salida del brazo (Simulink®).

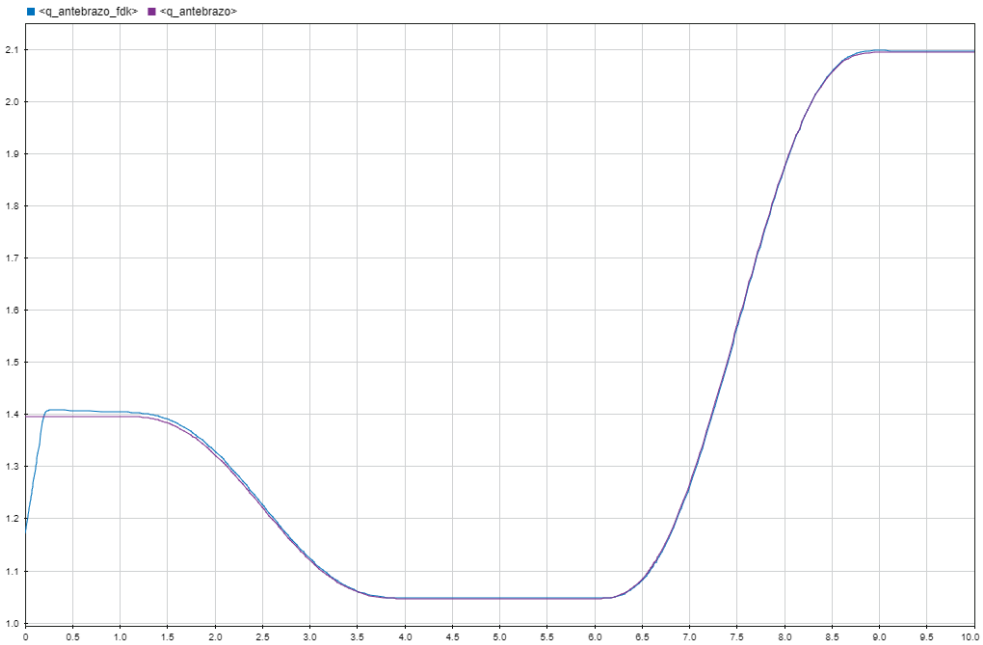


Figure 65. Señal de entrada y salida del antebrazo (Simulink®).

Como se observa las señales obtenidas son muy similares a las de la referencia, pero en cambios muy bruscos la señal no la sigue con total exactitud como se muestra en el sobre pico inicial de la gráfica de antebrazo, el cual es algo muy difícil de seguir en un lapso de tiempo muy corto que en este caso ronda al rededor los 200 mili segundos, pero para aplicaciones en general con movimientos suaves el ajuste PID está bien.

4.4 Inconvenientes y recomendaciones

El principal conveniente que se presentó, fue con la implementación de control de posición y velocidad de los motores en el entorno de simulink en tiempo real, ya que la librería de Simulink Support Package for Arduino Hardware v18.1.1 y versiones anteriores no permiten realizar la lectura de más de dos encoders en la Arduino Mega, y la librería Legacy MATLAB and Simulink Support for Arduino que no tiene soporte actualmente (se utilizó la versión de Matlab 2015ª), dio problemas a la hora de correr el programa desde el arduino, lo cual se opta por utilizar el control de posición y velocidad, realizado en el IDE de arduino. Lo cual es más recomendable, ya que en este entorno se puede optimizar y mejorar la eficiencia del control y del código en general.

CAPÍTULO 5, CINEMÁTICA DIRECTA E INVERSA

Esta sección muestra la formulación de la cinemática directa e inversa, imprescindible para realizar control cinemático. El robot como se describe anteriormente consta de tres eslabones que requieren de su actuador para poder controlar la estructura. Adicionalmente cuenta con un eslabón más que siempre está paralelo a la base, y depende del movimiento de los eslabones brazo y antebrazo. Los cálculos de las cinemáticas se realizan a partir de un brazo de 3 grados de libertad como se muestra en la figura 66, ya que este es el número de actuadores que se utilizan. Por otro lado, el último eslabón hace parte del efector final y la longitud de este depende del tipo de herramienta que se instale.

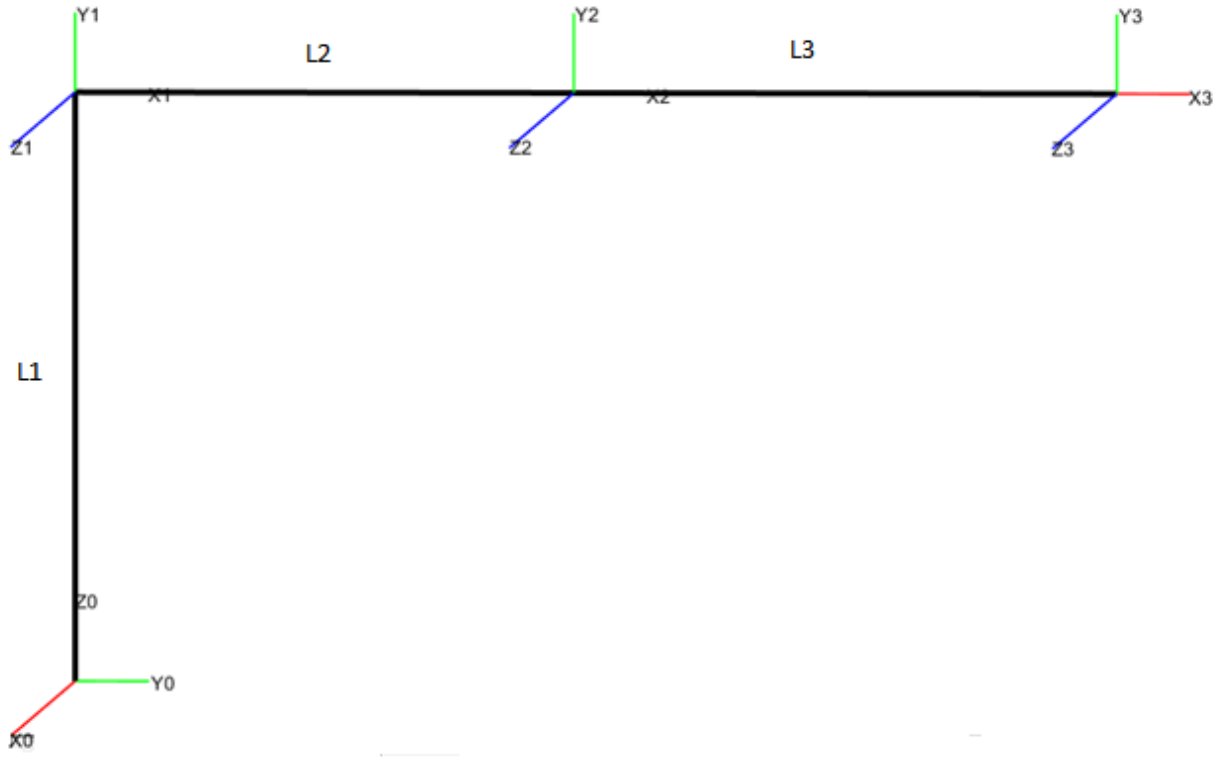


Figure 66. Inicio cinemático del brazo robótico.

5.1 Cinemática Directa

La cinemática directa consiste en determinar la posición y orientación del efector final del brazo robótico respecto al sistema de coordenadas, el problema cinemático directo se resuelve a partir de los parámetros de Denavit Hartenbert.

Table 1
Parámetros Denavit Hartenbert.

Ɵ	**d**	**a**	**α**
q1+pi/2	L1	0	pi/2
q2	0	L2	0
q3	0	L3	0

Parámetros para calcular la cinemática directa.

El parámetro **a** es referente a la distancia entre los ejes i e i-1, cual es el encargado de definir la longitud del eslabón, y el signo es determinado por la regla de la mano derecha. El parámetro **α,** es formado entre los ejes i e i-1 de las articulaciones. El parámetro **d** es la distancia desde el origen de un sistema articulación eslabón, hasta la intersección del eje Z y X del sistema anterior. El parámetro **Ɵ**, es un ángulo formado por el eje x con respecto al eje x-1.

La función en Matlab de la cinemática directa que se encuentra en la sección de anexos B, para poder dibujar el robot en forma de alambre a partir de las matrices de transformación homogénea, se realiza una función llamada dibujar robot (anexo C). La figura 67 muestra el robot en forma de alambre, esto con el motivo de verificar de forma visual la posición del robot.

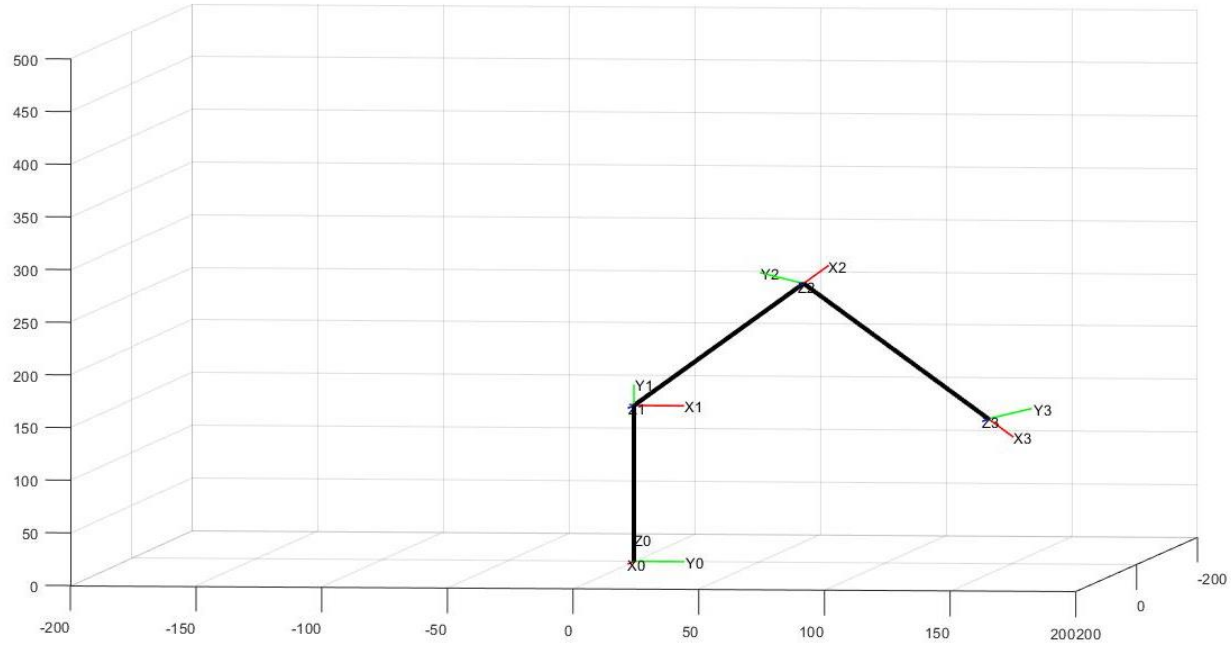


*Figure 67. Robot en alambres a partir de la cinemática directa con valores angulares q1=0, q2= pi/3, q3=-2*pi/3.*

Una vez verificado la formulación del problema cinemático directo, se procede a implementar en el entorno de Simulink® mediante un bloque de Matlab Function como se muestra en la figura 68.

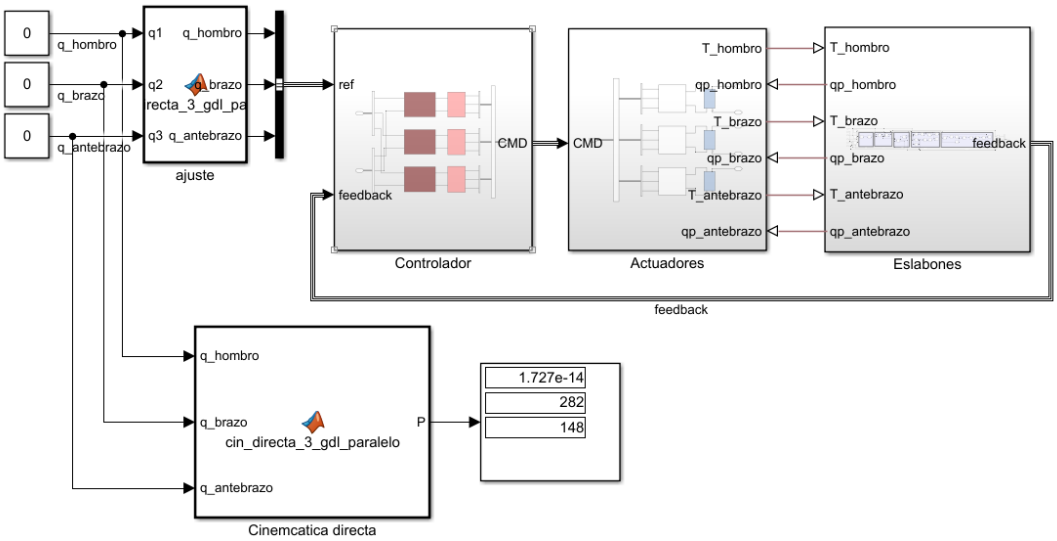


Figure 68. Cinemática directa en el entorno de simulink.

El bloque de ajuste agrega un valor angular de -pi/4 al antebrazo, y pi/2 al valor del brazo, esto con el fin de arreglar el desfase que existe con el modelo generado mediante simscape respeto al modelo de inicio cinemático del robot de la figura 66. La figura 69, ilustra el inicio cinemático del robot del modelo generado en simscape.

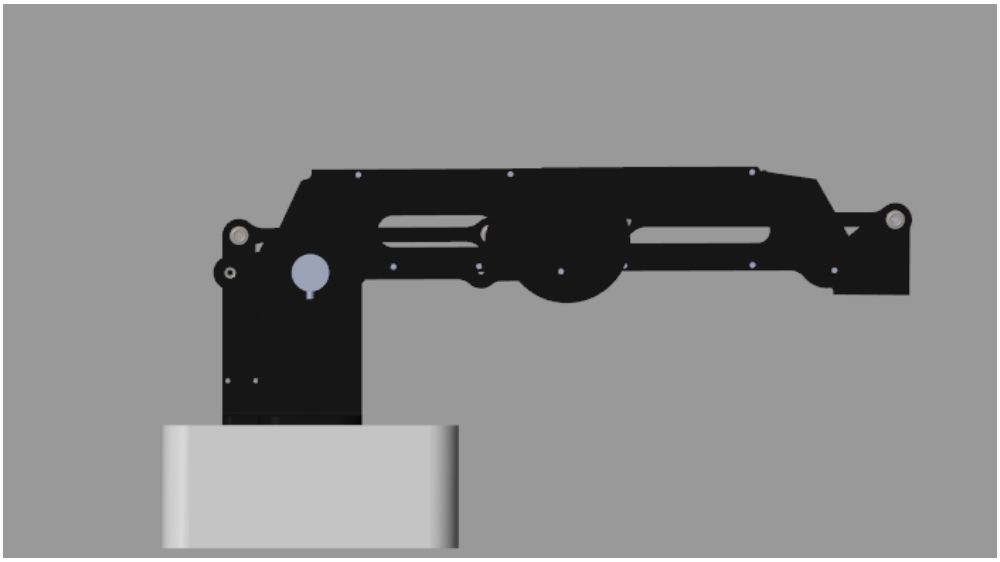


Figure 69. Robot en el origen cinemático.

Algo a mencionar, es que este tipo de robot la perspectiva de los movimientos angulares representada en el eslabón antebrazo, no es igual al de los robots antropomórficos convencionales, colocando de ejemplo los valores de q1=0, q2=pi/2, q3=0, la visualización de un robot antropomórfico convencional seria de la siguiente manera (figura 70).

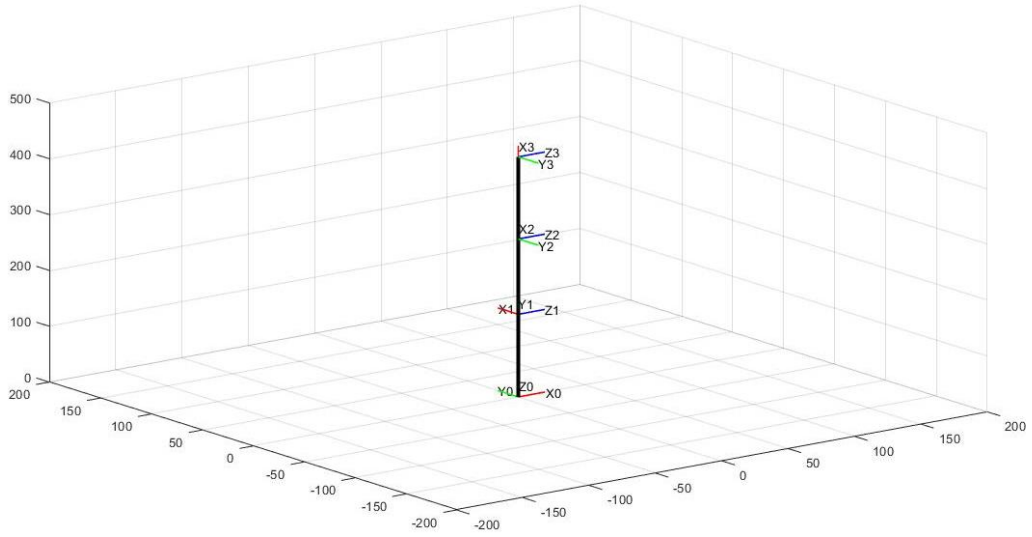


Figure 70. Robot antropomórfico convencional, con los valores angulares 0, pi/2, 0.

En el caso del modelo en simsacape (figura 71), la visualización es de la siguiente manera.

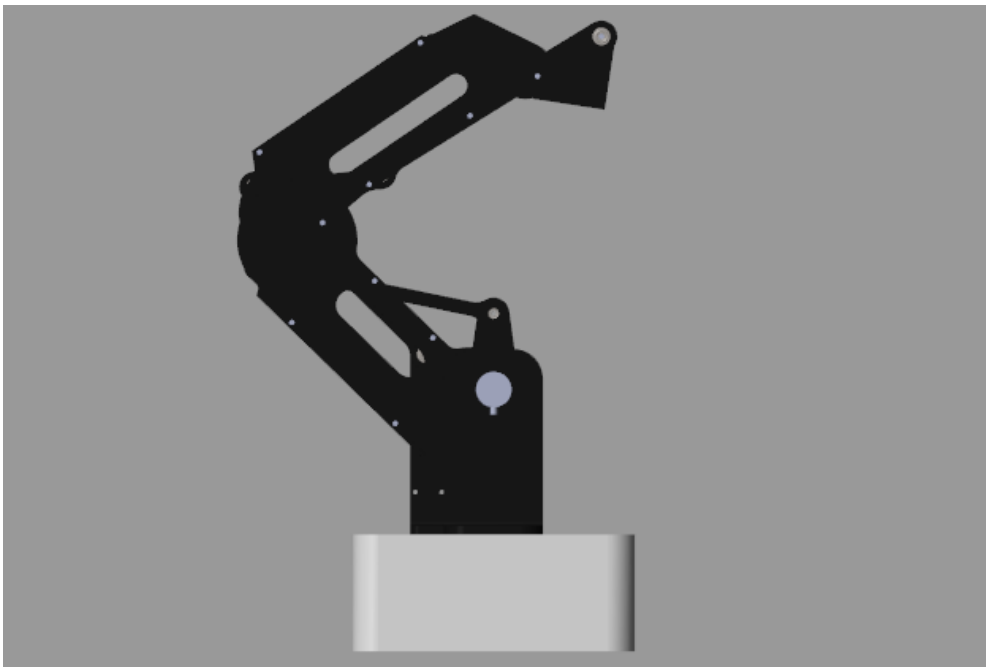


Figure 71. Robot con los valores angulares 0, pi/2, 0.

El desfase es debido a que el modelo simscape al inicio cuenta con un valor q2 de p/4 y q3 de -pi/2, por esta razón se utiliza un ajuste de q2 de –pi/4 y q3 de pi/2 para que el inicio del robot coincida el modelo en alambres.

5.2 Cinemática Inversa

La cinemática inversa, consiste en determinar la configuración que debe adoptar el brazo robótico para una posición y orientación del efector final, el problema cinemático inverso se realiza por medio de métodos geométricos. En primer lugar, se muestra la vista del brazo robótico desde sistema de coordenadas en Zo, como lo indica la figura 65.

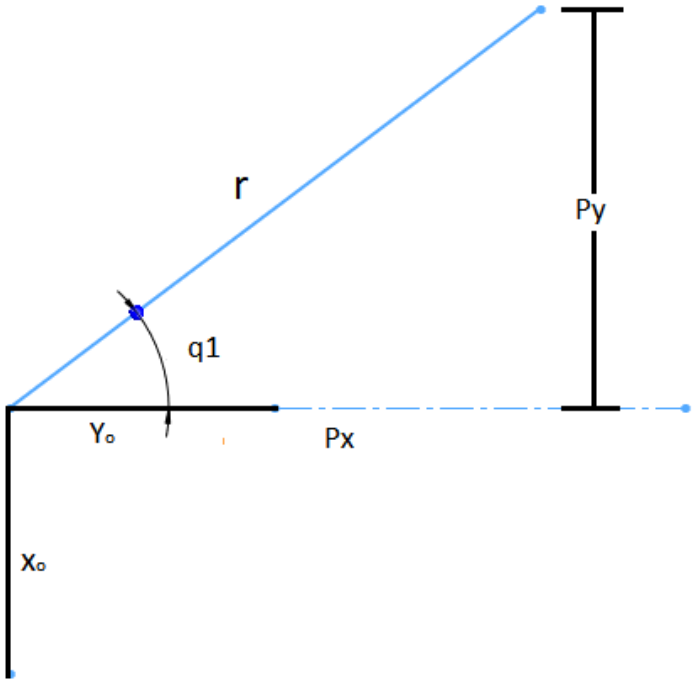


Figure 72. Brazo visto desde Zo.

A continuación, se obtienen las respectivas ecuaciones, observando el robot desde Zo, donde.

$$q_1 = ATAN2(-P_x, P_y); \quad \text{Ec. 20}$$

$$r = sqtr((-P_x)^2 + (P_y)^2); \quad \text{Ec. 21}$$

La ecuación 20 obtiene el valor articular del Angulo q_1, la ecuación 21 encuentra la distancia del efector final respecto al sistema de coordenadas 1.

Después de que se obtiene las expresiones 20 y 21 se cambia de vista el brazo robótico en z1, como indica la figura73.

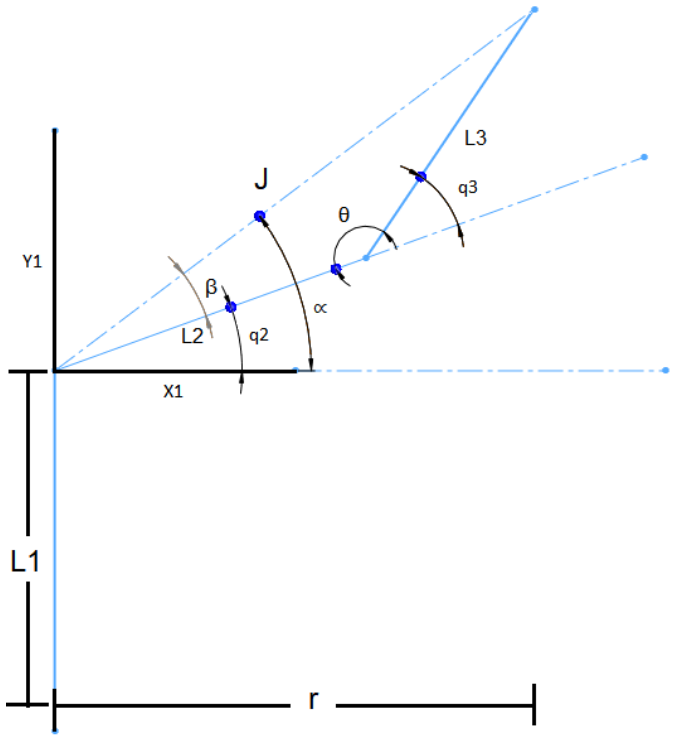


Figure 73. Brazo visto desde Z1.

Por medios geométricos se obtienen las siguientes expresiones.

$$J = sqtr(r^2 + (P_z - L_1)^2); \quad \text{Ec. 22}$$

$$J^2 = {L_2}^2 + {L_3}^2 - 2L_2L_3\cos(\theta); \quad \text{Ec. 23, } Donde, \theta = 180 - q3 \quad \text{Ec. 24}$$

Reemplazando el valor de θ (ecuación 24) en la ecuación de 23.

$$J^2 = {L_2}^2 + {L_3}^2 - 2L_2L_3\cos(180 - q3); \quad \text{Ec. 25}$$

$$\cos(180 - q3) = \cos(180)\cos(q3) - \sin(180)\sin(q3) = -\cos(q3); \quad \text{Ec. 25.a}$$

Aplicando la identidad geométrica de suma y resta de ángulos en coseno (25.a) y reemplazando en la ecuación 25 se obtiene la expresión 26.

$$J^2 = {L_2}^2 + {L_3}^2 + 2L_2L_3\cos(q3); \quad \text{Ec. 26}$$

Despejando cosq3,

$$\cos q3 = \frac{J^2 - {L_2}^2 - {L_3}^2}{2L_2L_3}; \quad \text{Ec. 27}$$

Aplicando la siguiente identidad trigonométrica se encuentra la expresión 28.

$$\sin q3 = codo * sqtr(1 - cos^2 q3); \quad \text{Ec. 28}$$

La expresión codo es un valor que nos indica si la solución es codo arriba, que en este caso tiene un valor de -1, o codo abajo con valor 1. Una vez se encuentren los valores de 27 y 28 se obtiene el valor angular de q3 con la expresión 29.

$$q_3 = ATAN2(sinq3 + cosq3); \quad \text{Ec. 29}$$

Ahora que se tiene el valor q3 se obtiene el valor de q2 con las siguientes expresiones.

$$q_2 = (\propto - \beta); \quad \text{Ec. 30}$$

$$\propto = ATAN2(Pz - L1, r); \quad \text{Ec. 31}$$

$$\beta = ATAN2(L3sinq3, L2 + L3cosq3); \quad \text{Ec. 32}$$

Una vez se encuentra los valores de las ecuaciones 31 y 32 se obtiene el valor de q2. Para más información el código se encuentra en anexo D.

Para corroborar la cinemática inversa de forma visual, se crea un pequeño programa que, mediante la posición obtenida en el efector final, se ingresa a la cinemática inversa, y los valores angulares obtenido se ingresan a la cinemática directa, y mediante la función dibujar robot imprime ambas imágenes.

```
% HOME
clear, clc
% Ejemplo
q = [0, pi/3, -2*pi/3];   codo = -1; % codo abajo = 1 ; codo arriba = -1
% directa
PARALELO = cin_directa_3_gdl_paralelo(q);
figure(1);
dibujar_robot(PARALELO); grid on; rotate3d
% Inversa
qi = Cinematica_inversa_3_gdl_paralelo(codo,PARALELO.A03(1:3,4))
PARALELO2 = cin_directa_3_gdl_paralelo(qi);
figure(2);
dibujar_robot(PARALELO2); grid on; rotate3d
```

A continuación, se muestra la figura del robot de alambres, a partir de la cinemática inversa, y como vemos es exactamente igual a la imagen a partir de la cinemática directa (imagen de la izquierda).

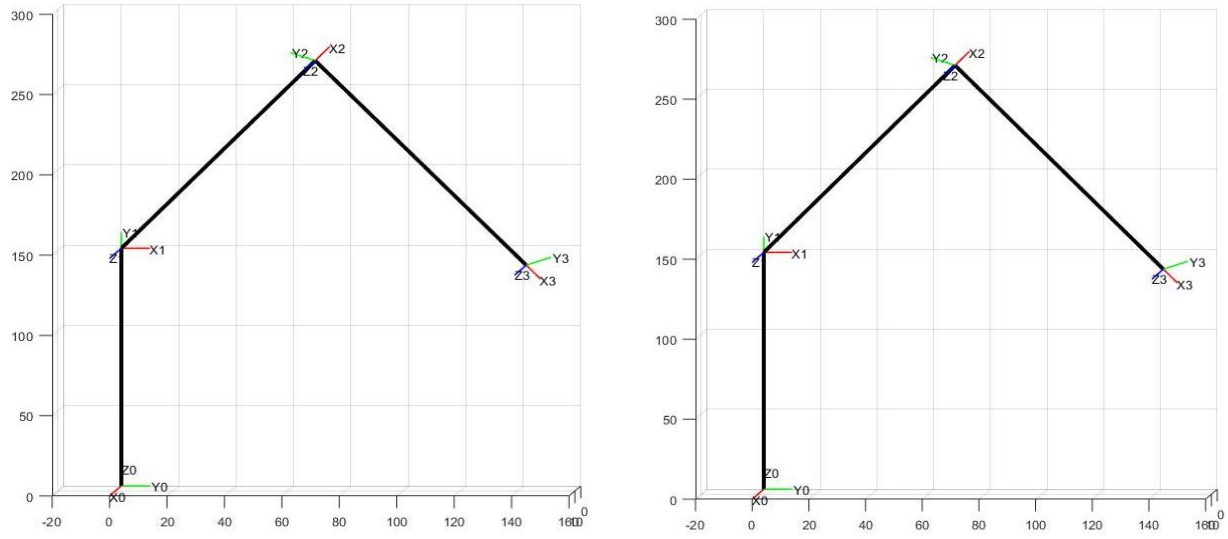


Figure 74. Robot en alambres corroboración cinemática inversa.

Una vez realizadas las comprobaciones, se procede a implementar la función en un bloque de simulink para ver su comportamiento.

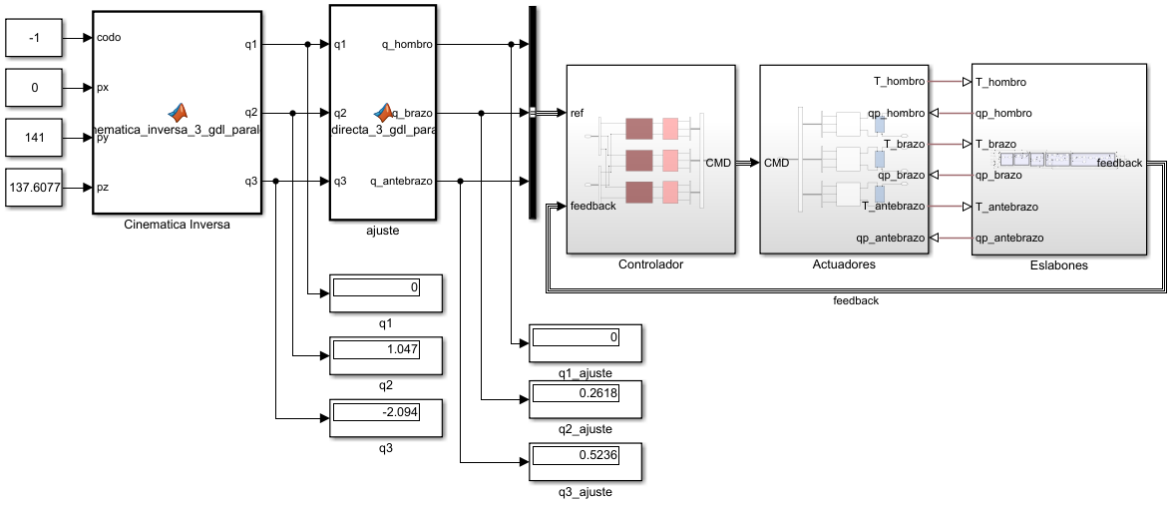


Figure 75. Cinemática inversa en Simulink®.

El posicionamiento del robot en un entorno virtual es el siguiente, y como vemos, es muy similar a las figuras de alambre obtenidas anteriormente.

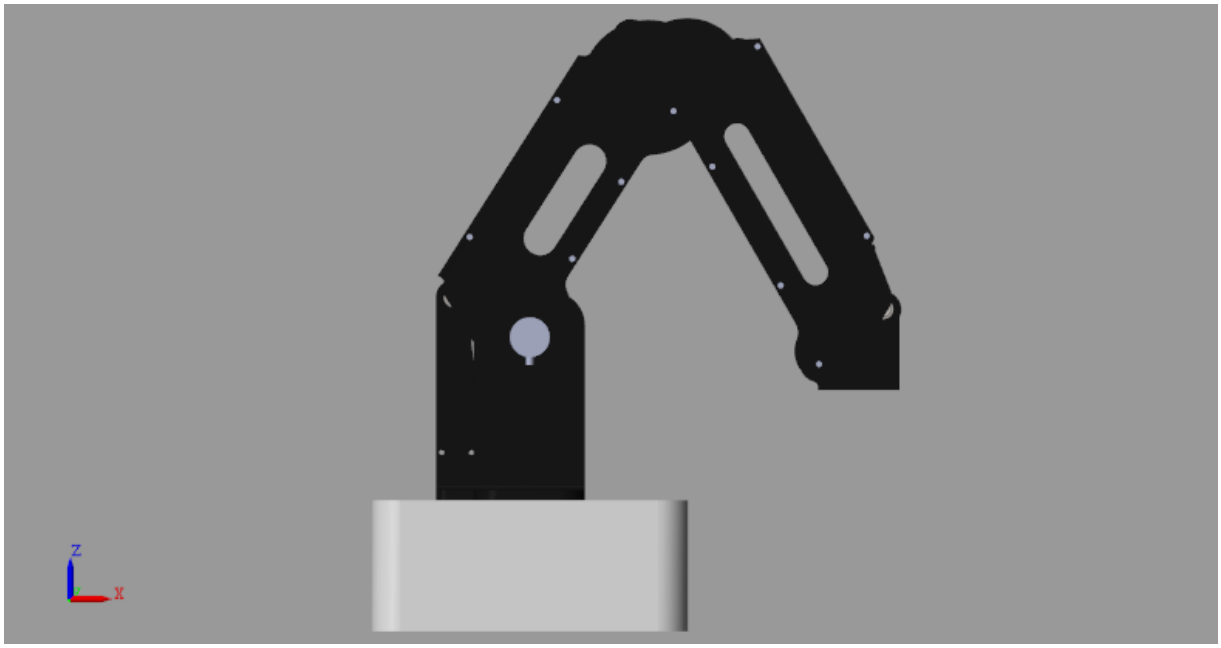


Figure 76. Robot con cinemática inversa posición xyz en 0, 141, 137,8 (Simscape).

Las figuras 77, 78 y 79 representa los valores de referencia de hombro, brazo y antebrazo versus los valores de salida obtenidos, indicándonos el tiempo que tardan en alcanzar la referencia.

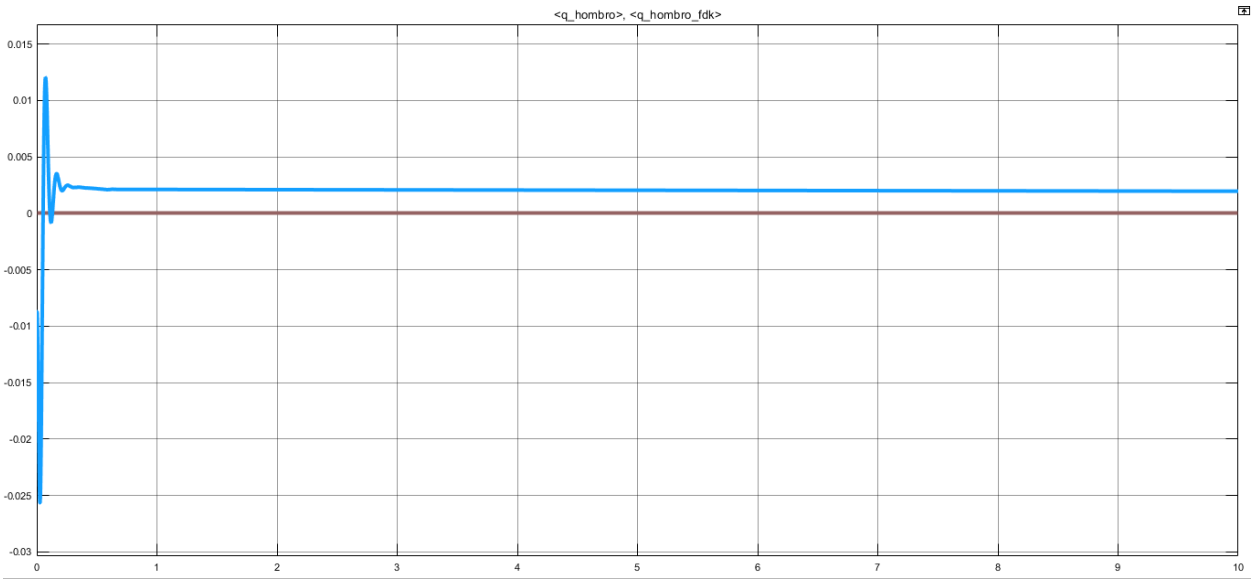


Figure 77. Referencia vs valor de salida del hombro.

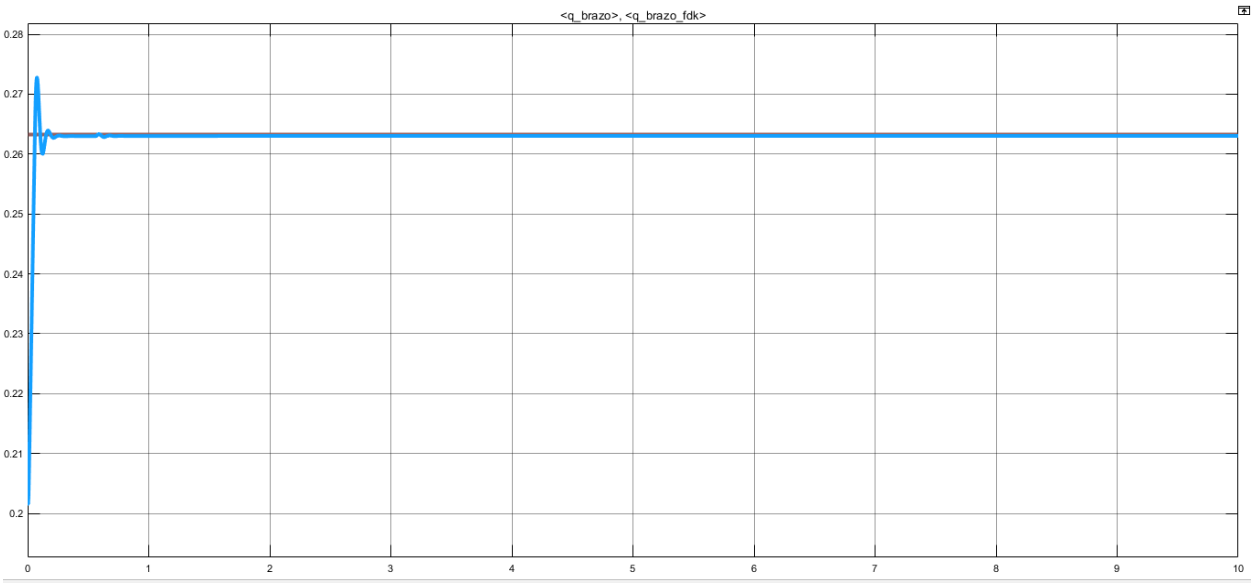


Figure 78. Referencia vs valor de salida del brazo.

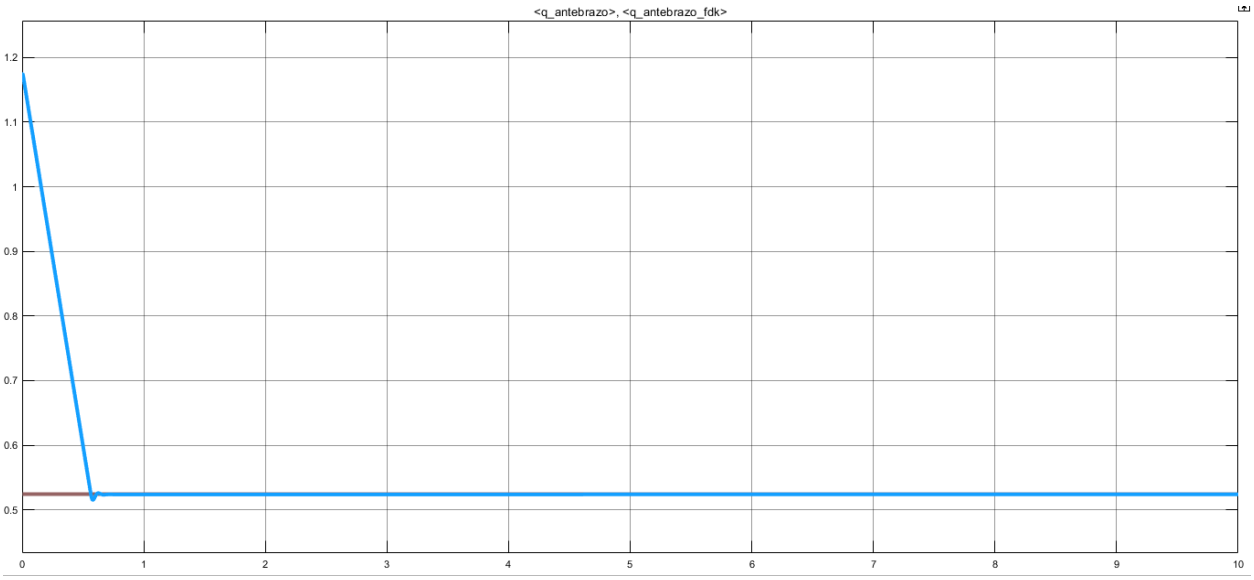


Figure 79. Referencia vs valor de salida del antebrazo.

Como se ilustran es la graficas de posición de hombro, brazo y antebrazo por medio de la cinemática inversa, se aprecia un pequeño sobre pico en los movimientos no tan bruscos como la figura 77 y 79, pero movimientos de mayor longitud y rapidez se genera un sobre pico que es poco apreciable, principalmente debido a altos valores de inercia, lo cual se bebe realizar una mejor sintonización, dependiendo la aplicación.

5.3 Inconvenientes y recomendaciones

El inconveniente que se presentó en esta etapa es con respecto origen cinemático y el origen del robot en el entorno de simulink, ya que este cuenta con un desfase en la articulación del brazo y antebrazo respecto al del origen cinemático, debido a que en momento que se exporta con simscape tiene cierta posición, lo cual para este caso la solución y lo que se recomienda es colocar el valor angular contrario para ajustar ese desfase.

CAPÍTULO 6, CONTROL CINEMÁTICO

Como se menciona anteriormente el objetivo del control cinemático es seguir cada articulación del robot a lo largo del tiempo para conseguir los objetivos fijados por el usuario, se clasifican en trayectorias punto a punto, y trayectorias continuas. Para el control cinemático en el torno de simulación de simulink se utiliza el diagrama de la figura 80.

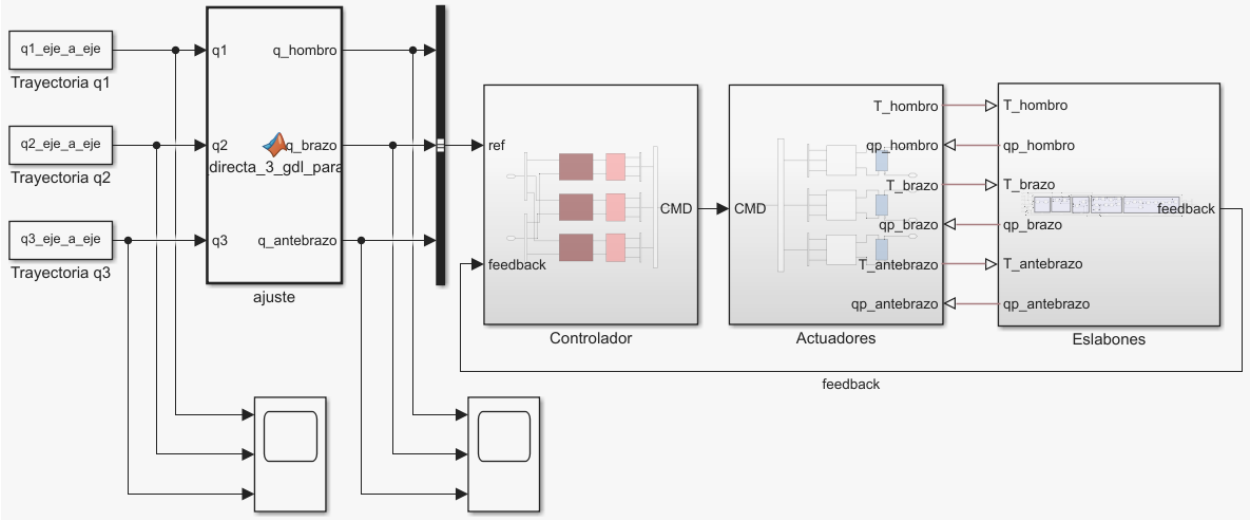


Figure 80. Modelo del robot en Simscape (referencia desde workspace).

La referencia ingresada es proveniente del workspace de Matlab®, se recurre a este método ya que Matlab en las version 2018ª la cual se utiliza en el proyecto, y versiones anteriores no soportan la generación de señales tipo times series en el bloque de funciones de simulink, por lo que se opta ejecutar los códigos de control cinemático en el editor de Matlab®.

6.1 Trayectorias punto a punto

Este tipo de trayectoria no importa el camino que tome el efector final del robot, sino que alcance el punto designado, se utiliza el interpolador trapezoidal (para más información ver anexo E) para poder calcular las trayectorias de las articulaciones, los tipos que se implementan en este proyecto son; movimiento eje a eje, movimiento simultaneo de ejes, y movimiento coordinado.

6.1.1 Movimiento eje a eje

El movimiento eje a eje, lo que hace es mover una articulación cada vez, hasta alcanzar el punto final designado, prácticamente el tiempo que tarda el ciclo es la suma del número de articulaciones

que se muevan. El diagrama de flujo (figura 81) explica de forma breve cómo funciona el código para la generación del movimiento eje a eje.

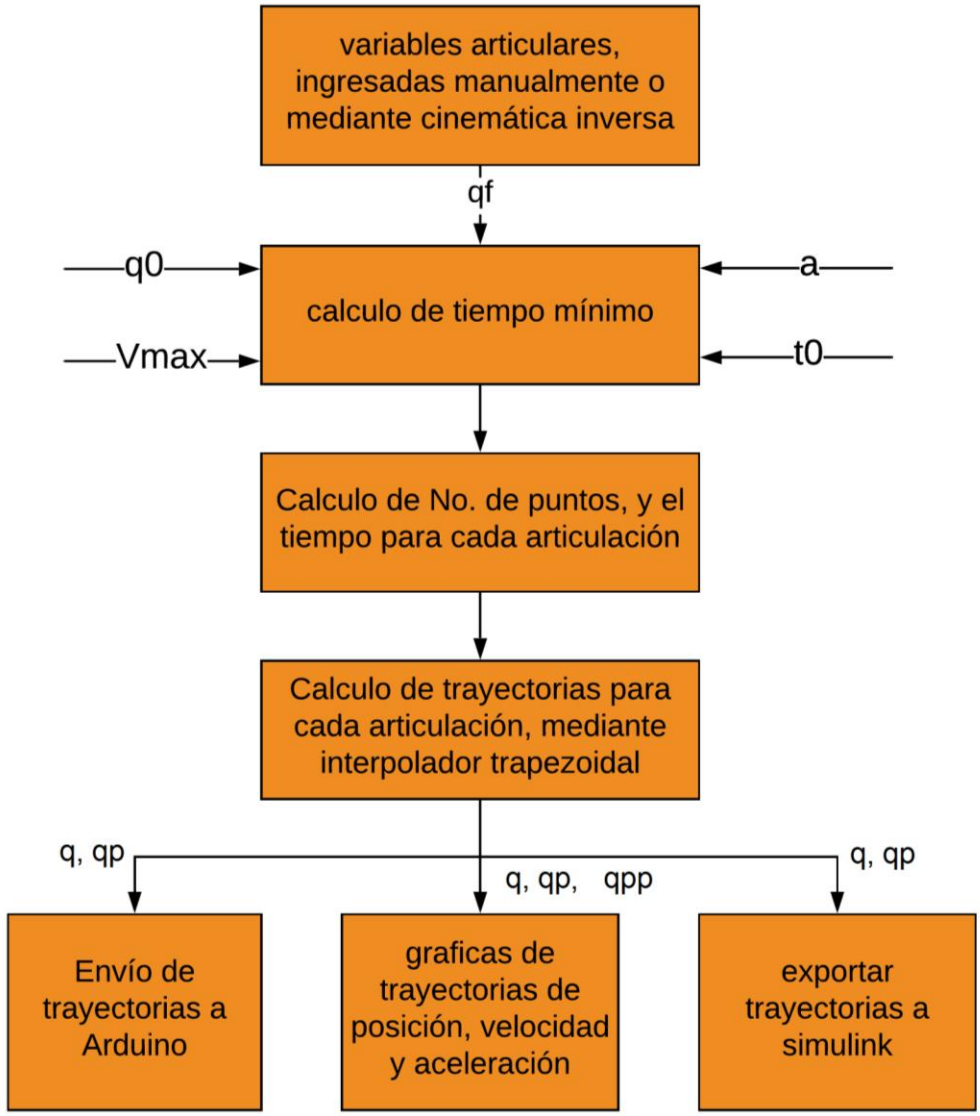


Figure 81. Diagrama de flujo para la explicación del movimiento eje a eje.

Como se indica en el diagrama de flujo el principio de funcionamiento no es complejo y los valores angulares se puede obtener mediante la cinemática inversa si lo requiere la aplicación, como también asignar esos valores de forma manual, para mayor detalle el código se encuentra en el anexo F. Se realiza una prueba con los valores angulares **q1=pi, q2=pi/3, y q3=-pi/3**. La figura 82 y 83 muestra una comparación del modelo obtenido en alambres y en simscape, con el objetico de corroborar los valores obtenidos por el código de movimiento aje a eje.

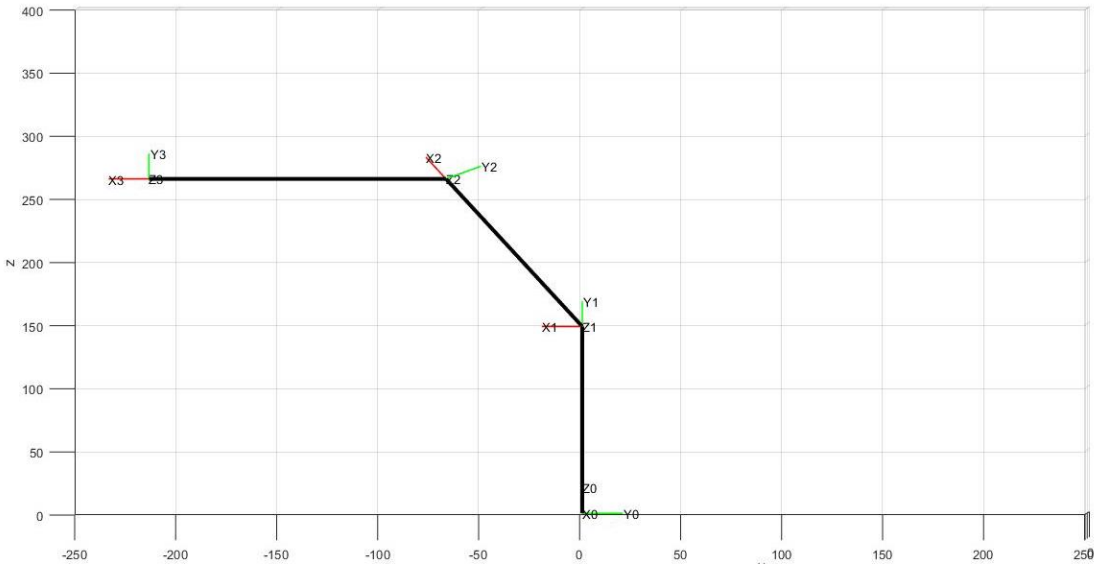


Figure 82. Posición del efector final del robot en alambres, mediante el movimiento eje a eje (q1=pi, q2=pi/3, y q3=-pi/3).

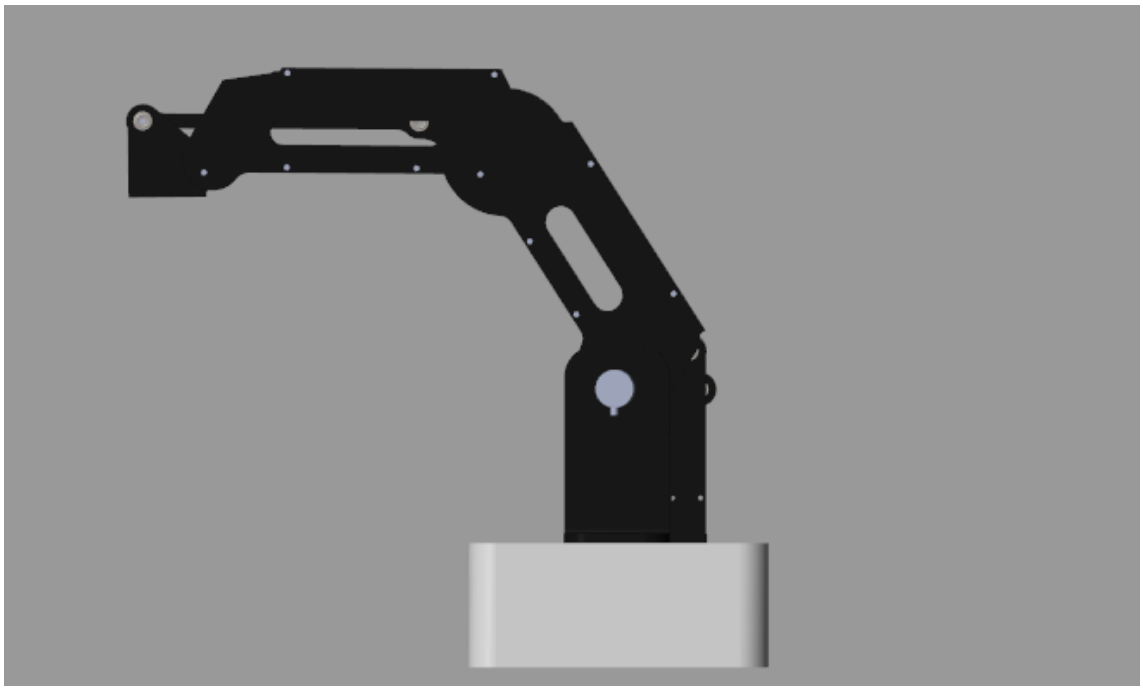


Figure 83. Posición del efector final del robot en Simscape, mediante el movimiento eje a eje (q1=pi, q2=pi/3, y q3=-pi/3).

Como se observa en las gráficas, las posiciones de los dos modelos de robot son coincidentes con los valores ingresados, al igual que sus movimientos en las articulaciones.

las gráficas de posición, velocidad y aceleración para cada articulación, que se obtienen para esta prueba son las siguientes (figuras 84, 85 y 86).

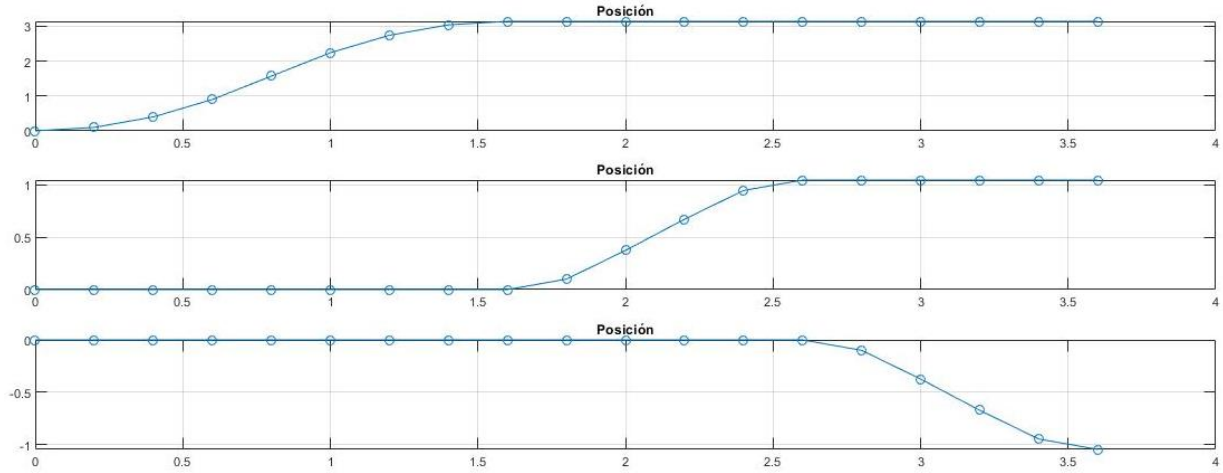


Figure 84. Trayectoria eje a eje graficas de posición.

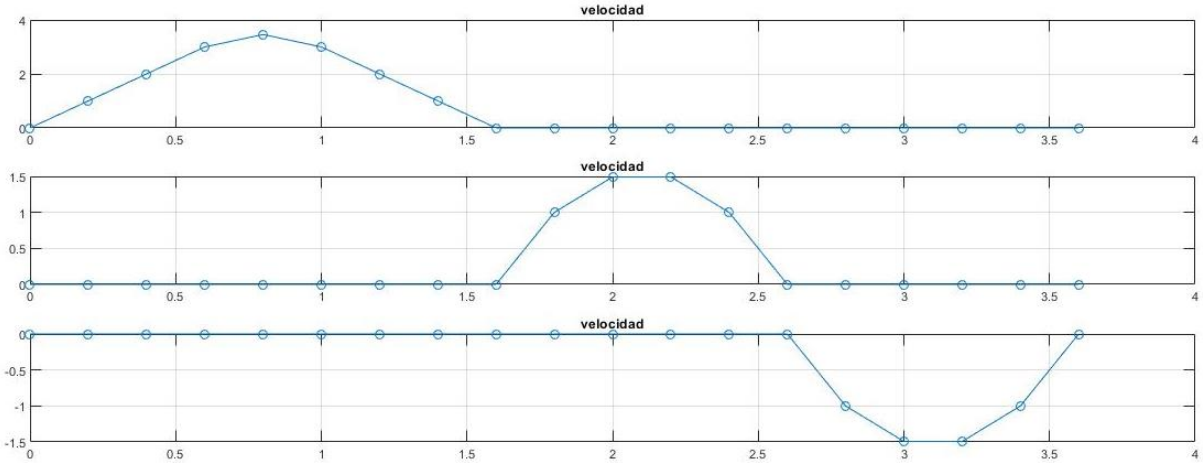


Figure 85. Trayectoria eje a eje graficas de velocidad.

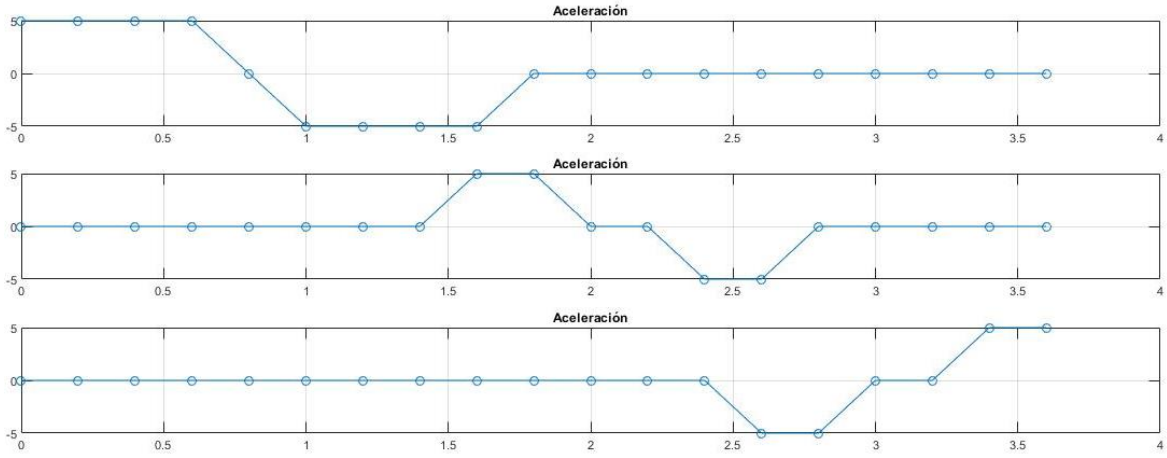


Figure 86. Trayectoria eje a eje graficas de aceleración.

Como se puede notar en las gráficas posición, velocidad y aceleración de las articulaciones, el número de puntos y su tiempo son iguales en las trayectorias entre cada articulación, y evidentemente este movimiento es el que más tarda de los tres tipos trayectorias punto a punto, expuestos en este proyecto, ya que solo se mueve un eslabón a la vez.

6.1.2. Movimiento simultáneo de ejes

En el movimiento simultaneo de ejes, los ejes comienzan a la vez, pero cada eje termina cuando puede, el diagrama de flujo explica el funcionamiento de forma fácil como funciona el código implementado que se encuentra en el anexo G.

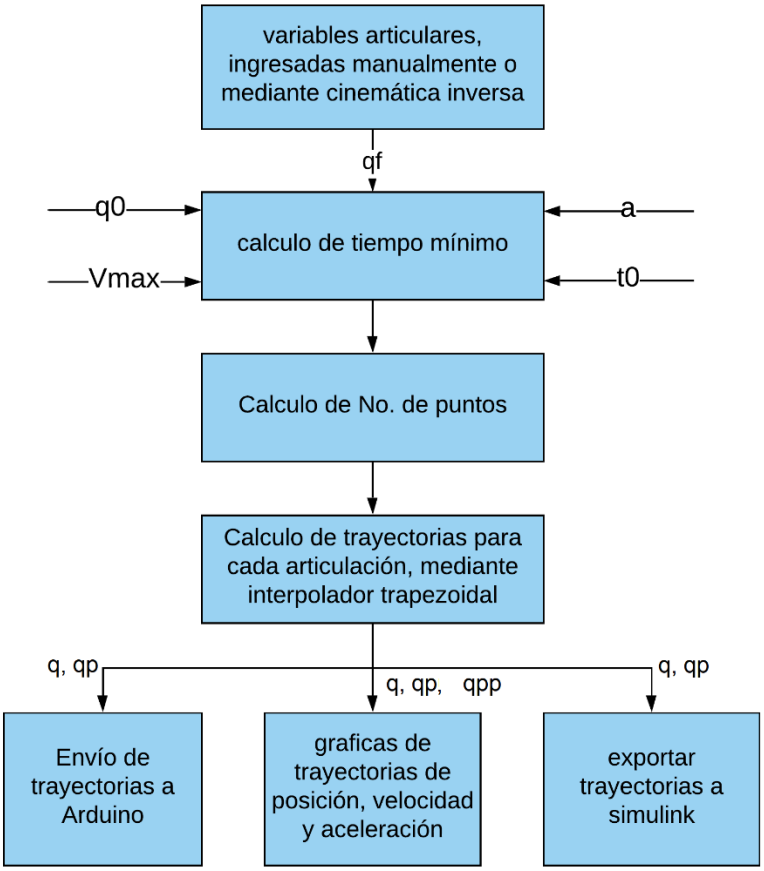


Figure 87. Diagrama de flujo para la explicación del movimiento de ejes simultáneos.

El principio de funcionamiento del código es muy similar al código de movimiento eje a eje, la diferencia radica en que el movimiento eje a eje se tiene que calcular el tiempo del inicio de la trayectoria hasta que termina, y el tiempo donde se empieza a mover la articulación, en cambio el movimiento simultaneo de ejes no se tiene en cuenta lo dicho, ya que todos los ejes inicializan a

la vez y terminan cuando pueden. Se ingresa los mismos valores articulares del movimiento eje a eje, y se corrobora con las posiciones obtenidas con el modelo en alambres y simscape.

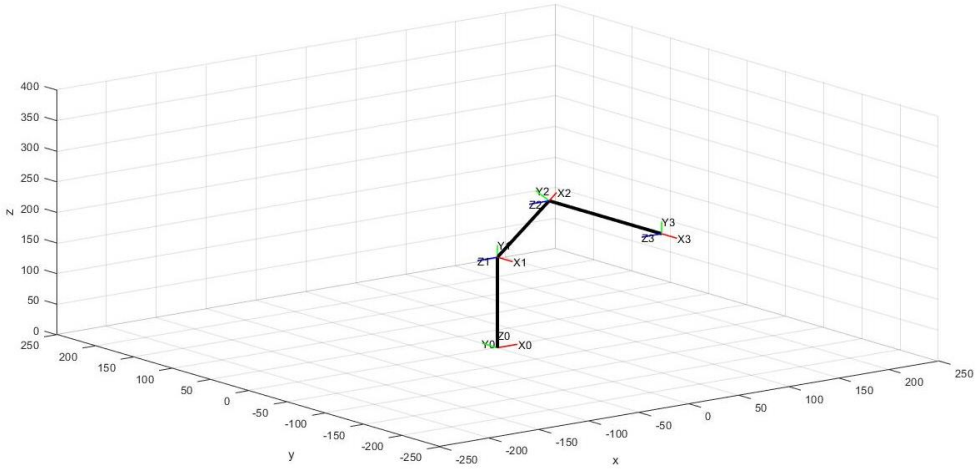


Figure 88. Posición del efector final del robot en alambres, mediante el movimiento simultáneo de ejes (q1=pi, q2=pi/3, y q3=-pi/3).

Figure 89. Posición del efector final del robot en Simscape, mediante el movimiento simultáneo de ejes (q1=pi, q2=pi/3, y q3=-pi/3).

Como vemos en las gráficas del robot en alambres y simscape, el posicionamiento es el mismo al igual que sus movimientos, como también igual el posicionamiento de las figuras 82 y 83 del movimiento eje a eje, el cual las figuras están en diferente vista en el plano. Las figuras 90, 91 y 92, representan las trayectorias posición, velocidad y aceleración de cada articulación.

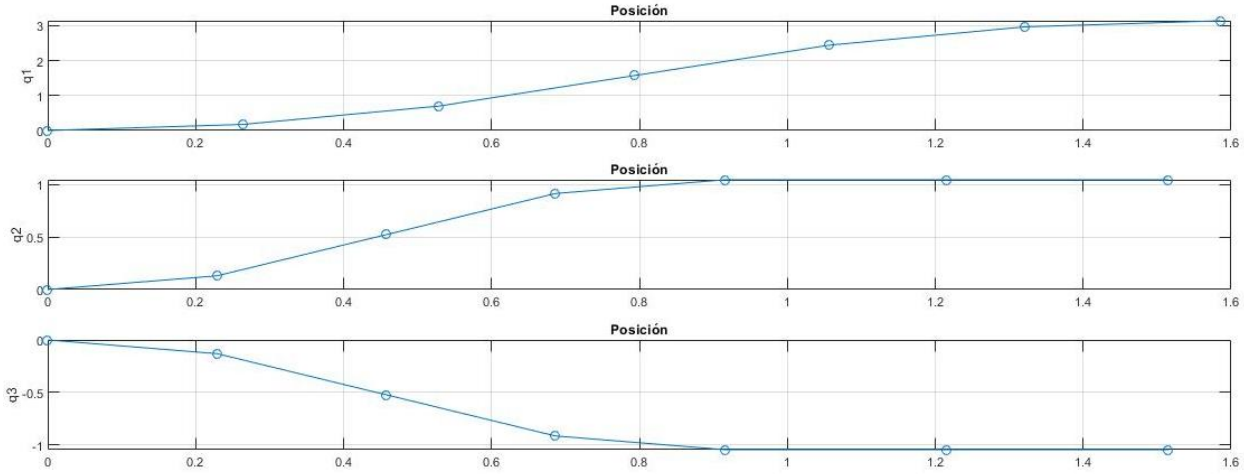


Figure 90. Trayectoria de movimiento simultaneo de ejes, graficas de posición.

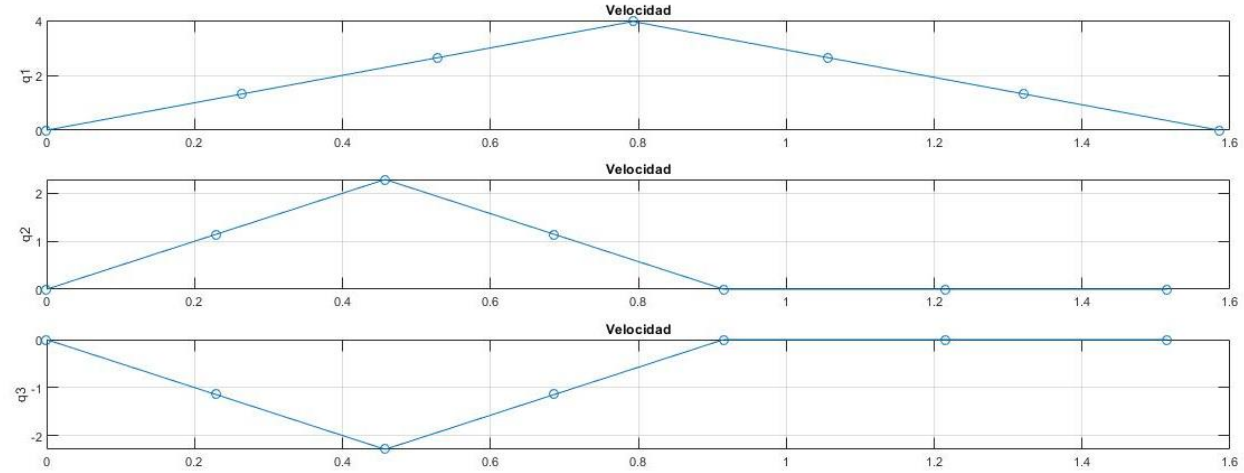


Figure 91. Trayectoria de movimiento simultaneo de ejes, graficas de velocidad.

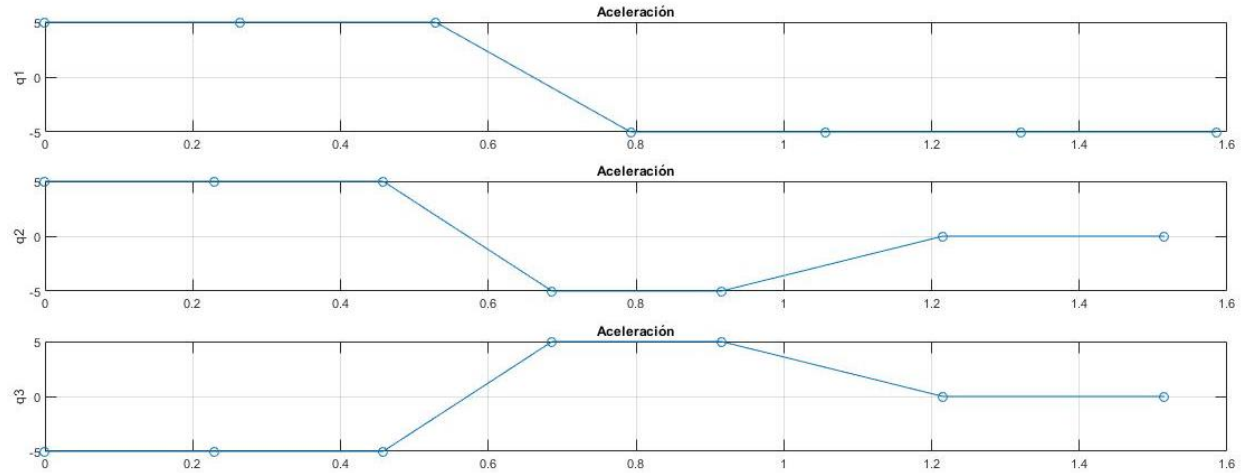


Figure 92. Trayectoria de movimiento simultaneo de ejes, graficas de aceleración.

Como se visualiza en las gráficas de posición, velocidad y aceleración de movimiento simultaneo de ejes, se tiene la misma cantidad de puntos entre la trayectoria, pero los tiempos no encajan a excepción de las dos últimas articulaciones, esto es debido a que tiene el mismo valor articular, pero con diferente signo. Las articulaciones terminan su movimiento en un tiempo diferente, a lo que conlleva que el movimiento del robot no termina hasta que el ultimo eje termine, esto implica altos requerimientos inútiles de velocidad y aceleración afectando los actuadores a largo plazo.

6.1.3. Movimiento coordinado

El movimiento coordinado a diferencia de los anteriores movimientos, las articulaciones comienzan y acaban a la vez, ralentizando los eslabones más rápidos. El diagrama de flujo de la figura 93 explica de forma breve el funcionamiento del código, para mayor información del código se encuentra en la sección de anexo H.

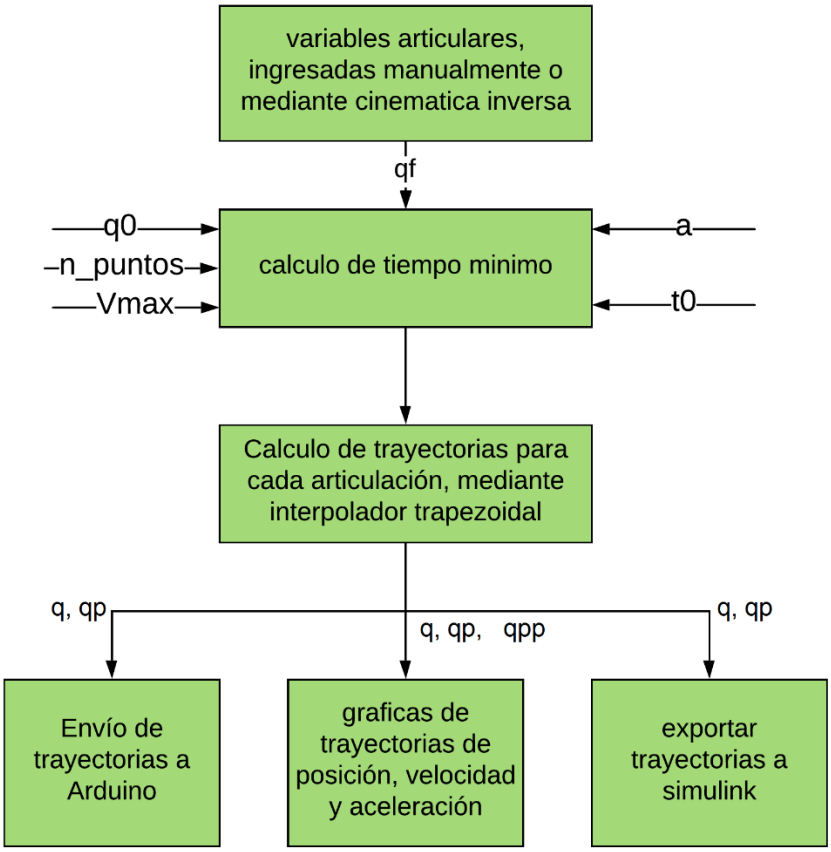


Figure 93. Diagrama de flujo para la explicación del movimiento coordinado.

A diferencia del movimiento eje a eje y simultaneo de ejes, el movimiento coordinado permite asignar el número de puntos que lleva la trayectoria, ofreciendo la ventaja de tener un movimiento más suave si la aplicación lo requiere asignando un mayor número de puntos. La figura 94 y 95, sirve para corroborar los valores angulares q1=pi, q2=pi/3, y q3=-pi/3 de referencia.

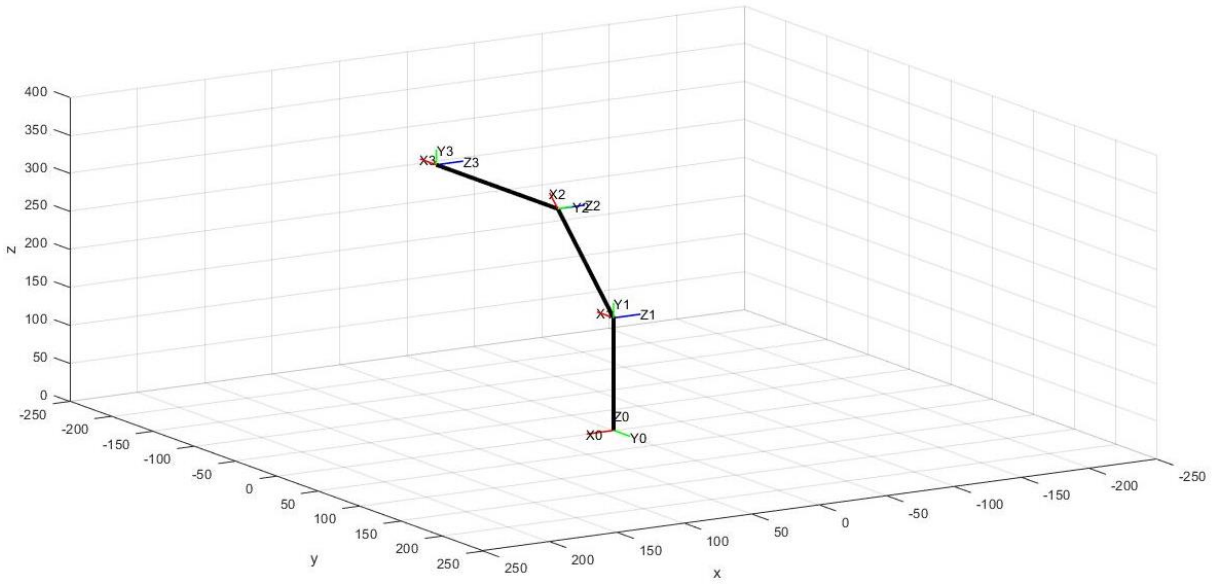


Figure 94. Posición del efector final del robot en alambres, mediante el movimiento coordinado (q1=pi, q2=pi/3, y q3=-pi/3).

Figure 95. Posición del efector final del robot en Simscape, mediante el movimiento coordinado (q1=pi, q2=pi/3, y q3=-pi/3).

Como se espera el posicionamiento del robot de alambre y en simscape son iguales, los valores angulares para los tres tipos de trayectorias son iguales, esto con el motivo de corroborar los tres códigos de trayectorias punto a punto implementados en este proyecto. Las figuras 96, 97 y 98, muestran las gráficas obtenidas de posición, velocidad y aceleración.

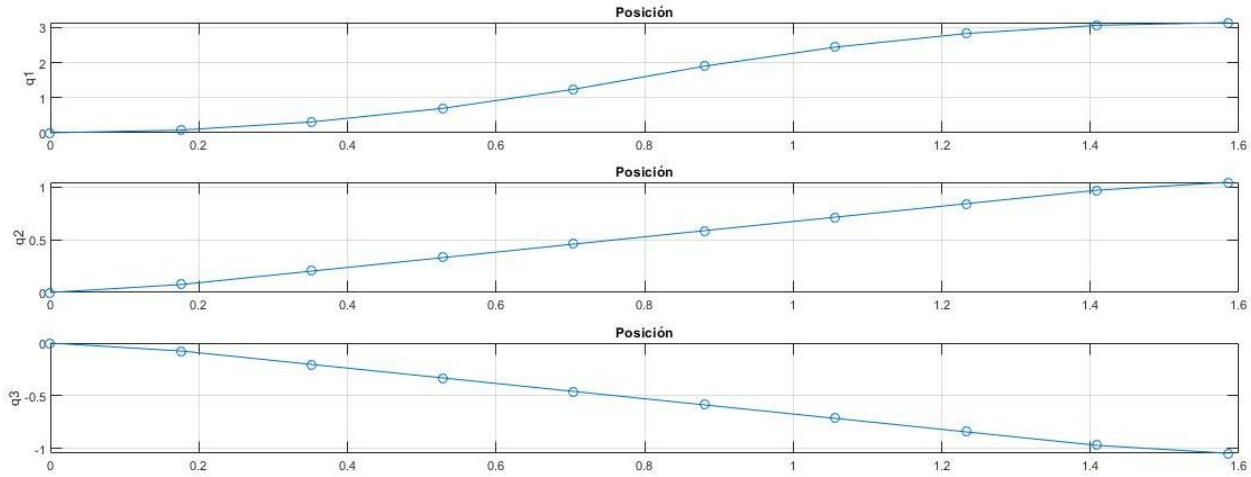


Figure 96. Trayectoria de movimiento coordinado, graficas de posición.

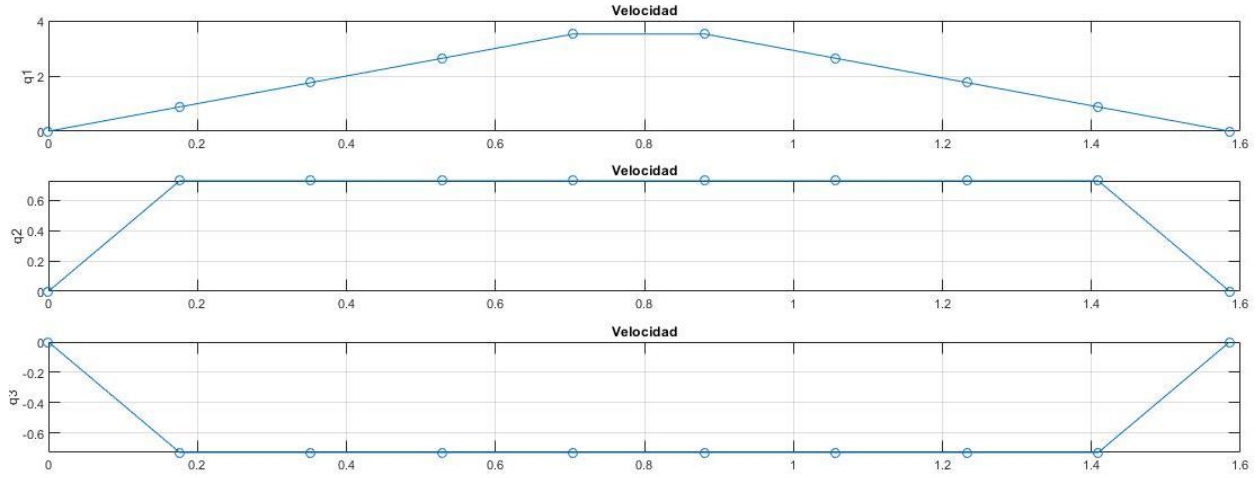


Figure 97. Trayectoria de movimiento coordinado, graficas de velocidad.

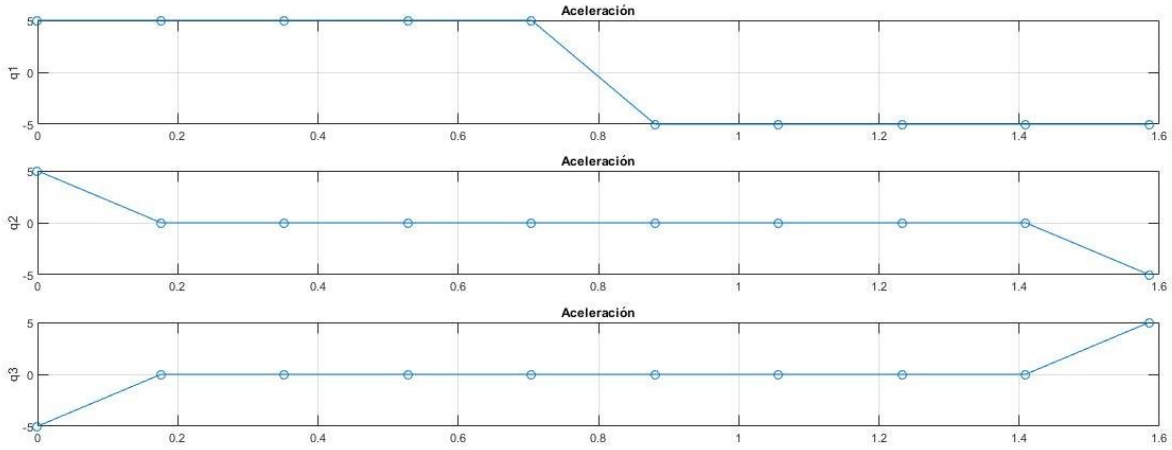


Figure 98. Trayectoria de movimiento coordinado, graficas de aceleración.

Como se muestra en la figura 96, 97 y 98 el número de puntos asignados (en este caso son 10), corresponde a las trayectorias de las tres articulaciones, coincidentes en el tiempo, esto se logra ralentizando las articulaciones más rápidas, de forma que las articulaciones terminen a la vez, evitando exigencia innecesaria de velocidad y aceleración.

6.2 Trayectorias continuas

A diferencia de la trayectoria punto a punto, se desea que el efector final del robot describa una trayectoria concreta y conocida, importando el camino que tome el efector final, esto es muy importante para tareas como soldadura, corte a laser, gravado, etc.

6.2.1 Trayectoria mediante splines cúbicos

Para este tipo de trayectoria se recurre al interpolador cubico, el cual une cada pareja de puntos con un polinomio de grado 3 que contiene cuatro parámetros, condiciones de contorno, posición y velocidad de comienzo fin. Para más información ver anexo I donde se encuentra los códigos del interpolador cubico. A continuación, se da una breve descripción mediante un diagrama de flujo (figura 99) del funcionamiento para generar la trayectoria mediante interpoladores cúbicos, el código que realiza esta acción se encuentra en el anexo J, debidamente explicado.

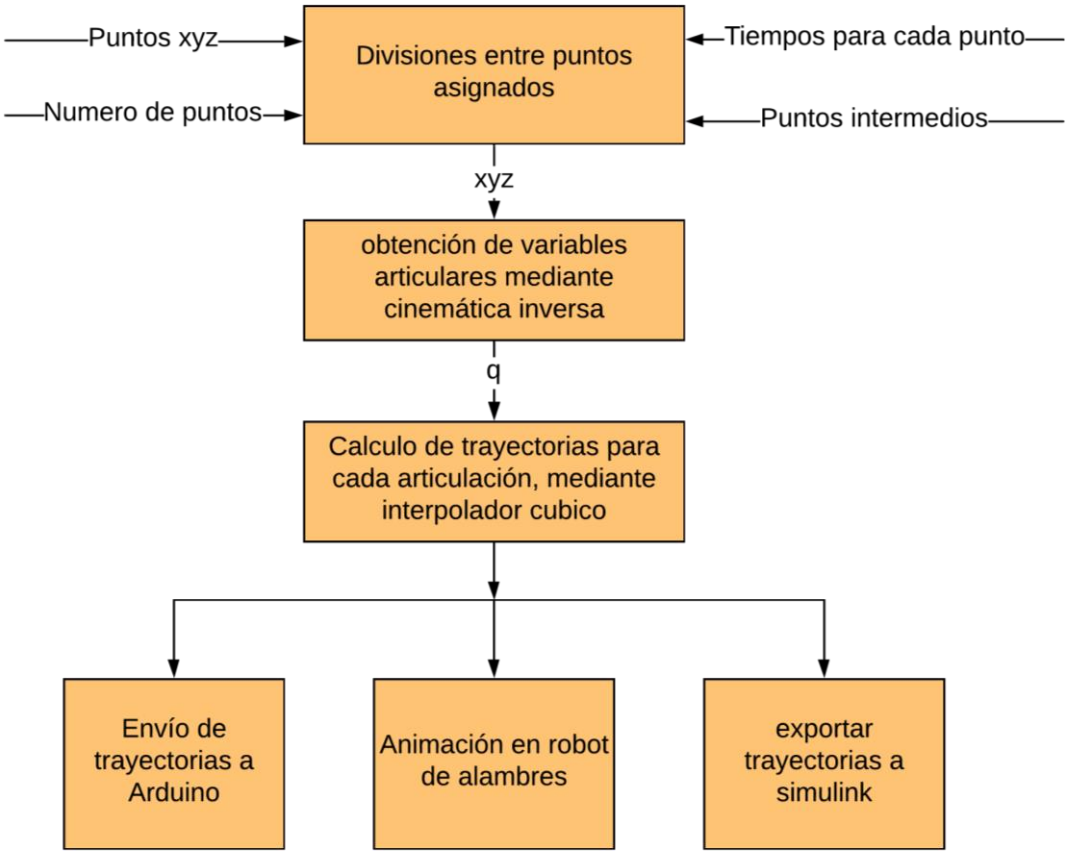


Figure 99. Diagrama de flujo para la explicación de la trayectoria continua.

Como se menciona anteriormente se usa en esta prueba la cinemática inversa para poder asignar la trayectoria de una figura de forma fácil, que en este caso es un cuadrado. Para poder formar el cuadrado se asignan los siguientes puntos de la tabla 2.

Tabla 2
Puntos para formar el cuadrado

Px	-100	100	100	-100	-100
Py	50	50	250	250	50
Pz	100	100	100	100	100

Puntos en plano, del efector final

Se asigna valor de 3 n_puntos al interpolador cubico, 15 divisiones intermedias entre cada punto asignado para forma la figura, esto con el fin de lograr más resolución, los resultados obtenidos se muestran en las siguientes figuras.

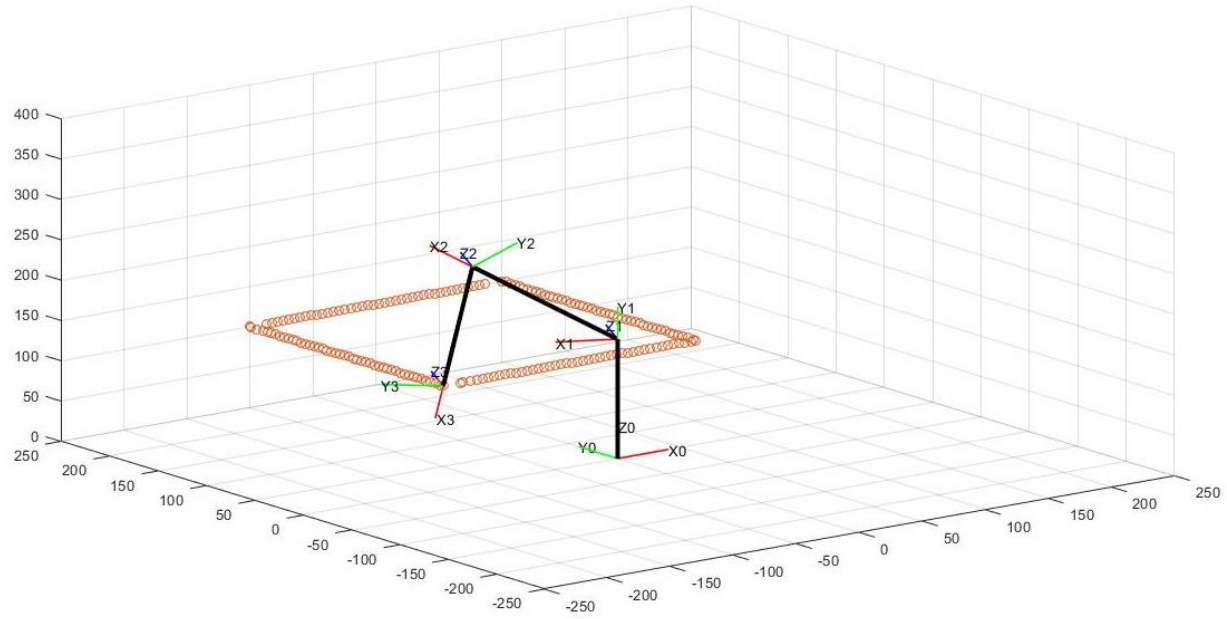


Figure 100. Trayectoria continua en forma de cuadrado.

La figura 100, muestra que los parámetros ingresados logran un buen resultado formando con bastante precisión el cuadrado. En una escala más detallada como lo muestra la figura 101, la trayectoria no es del todo perfecta, es por esto, que es necesario agregar los puntos que se requieran para mejorar la resolución de la misma, y como se observa, las zonas con mayor imperfección son las esquinas por tener forma de punta, ya que es más difícil de realizar la trayectoria en ese tipo de formas.

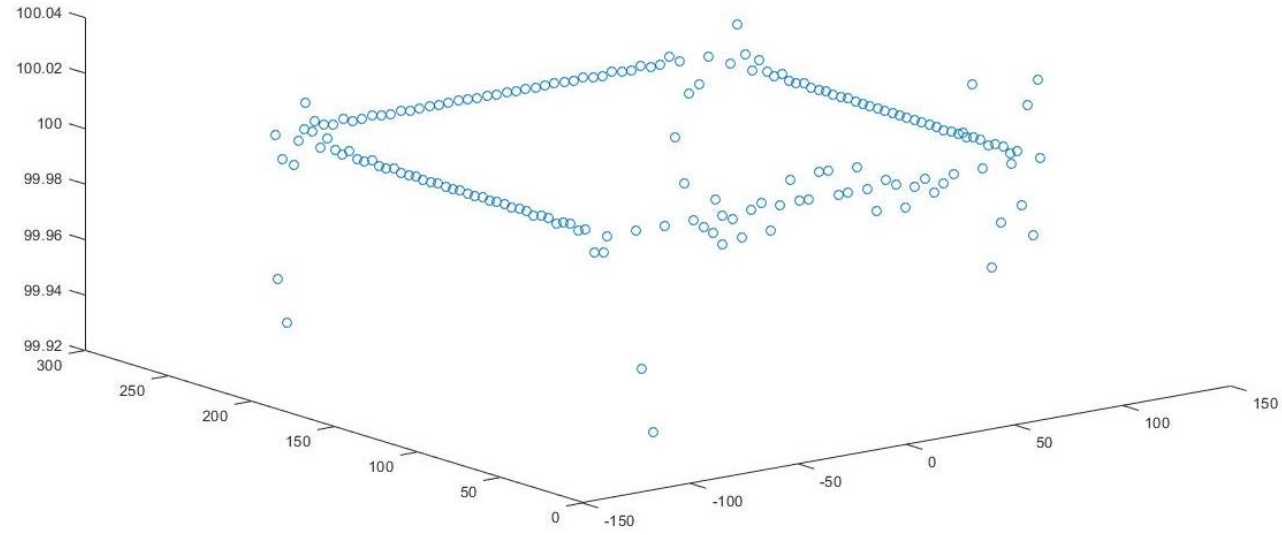


Figure 101. Trayectoria continua en forma de cuadrado, en escala más detallada ***(escala en mm).***

Las imperfeciones viendo a escala mas detallada no es mayor de un milimetro, lo que nos inidca que con esta configuración se logra muy buenos resultados y el coste computacional no es tan elevado. En el entorno de simulink las siguientes graficas (figura 102, 103 y 104) muestran el comportamiento que tiene cada articulación durante la trayectoria trazada.

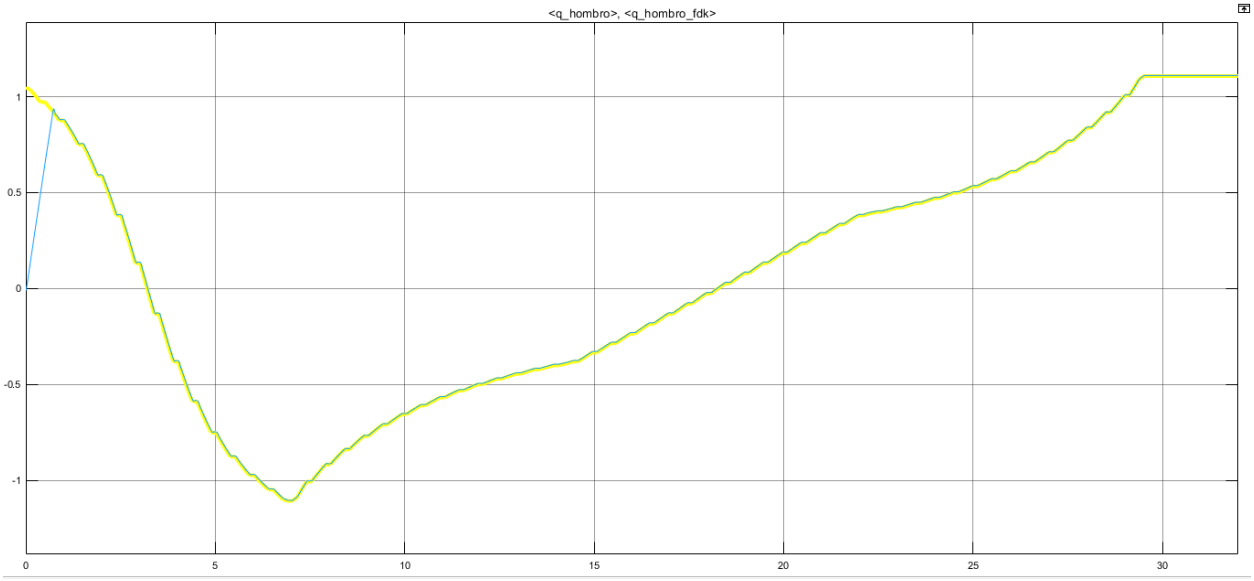


Figure 102. Movimiento de la articulación hombro vs la referencia.

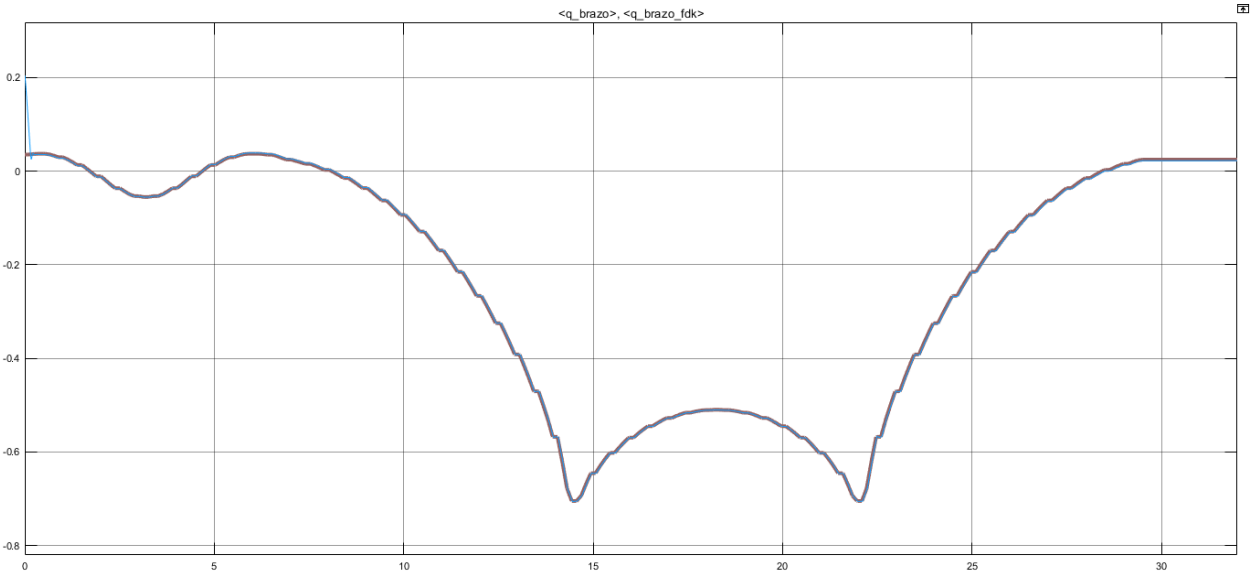


Figure 103. Movimiento de la articulación brazo vs la referencia.

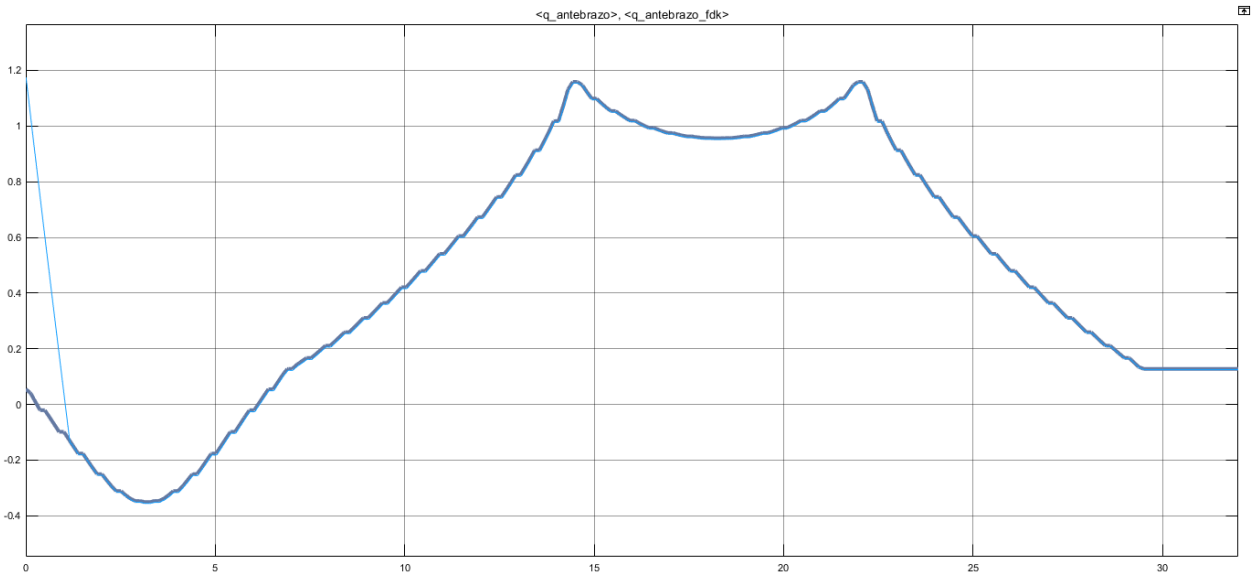


Figure 104. Movimiento de la articulación antebrazo vs la referencia.

Se presentan pequeños sobre picos que son poco apreciables al iniciar, debido a la inercia que genera la articulación para llegar al punto de partida de la trayectoria, esto es mayor si la posición donde se encuentra la articulación es más lejana, ya que este movimiento lo hace a bastante velocidad por ende se tiene una mayor inercia, y también depende de la eficiencia del controlador, pero en este caso funciona muy bien. Una vez se alcanza la referencia, las articulaciones siguen muy bien la trayectoria.

Para mejorar la precisión de la trayectoria, lo que se hace es asignar un mayor número de puntos, pero esta acción requiere más capacidad de procesamiento, y por ende más tiempo para realizar la trayectoria, por lo que se recomienda encontrar un equilibrio dependiendo la aplicación.

6.3 Inconvenientes y recomendaciones

El principal inconveniente que se presentó en esta etapa, como se menciona anteriormente, es en simulink, ya que no fue posible integrar todo en ese entorno, debido a la falta de soporte para utilizar variables objeto tipo times series en el bloque de Matlab function, ya que es muy necesaria para poder formar la señal a partir de las matrices obtenidas de la trayectoria, algo diferente cuando se utiliza solo cinemática ya que este solo manda un valor articular para formar la señal.

Otro inconveniente a mencionar, es de simulink al realizar la simulación del robot por un tiempo definido, cuando se crea la señal de la trayectoria por un intervalo de tiempo y tiene una forma

creciente o decreciente, y se simula a un tiempo mayor al generado, lo que sucede es que la señal obtenida con un tiempo mayor es de seguir el mismo patrón de incremento o decremento como lo muestra la trayectoria naranja de la figura 1.

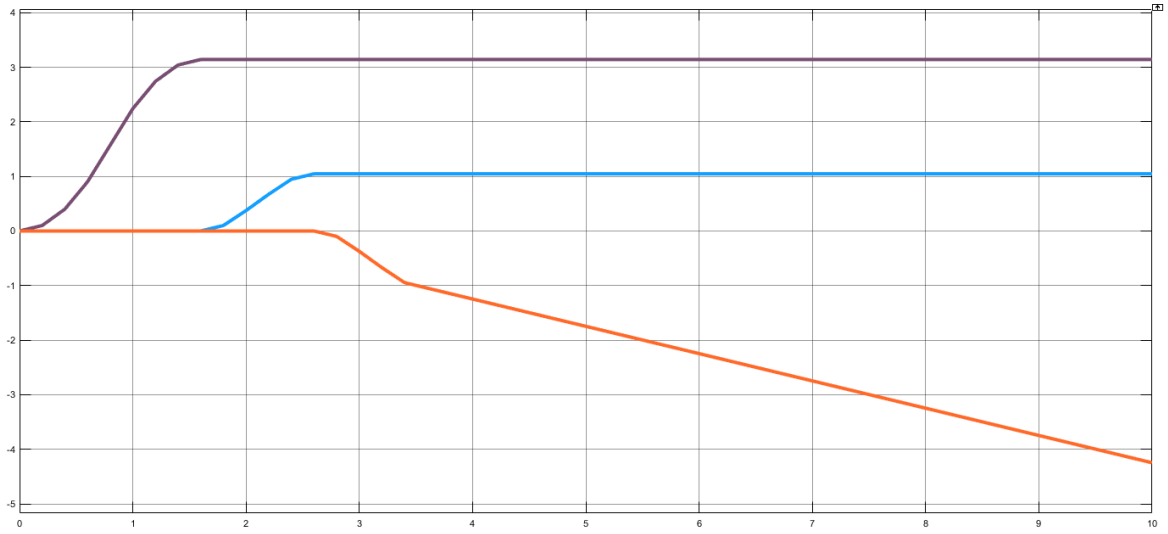


Figure 105. Sin ajuste de error de incremento.

Cabe recalcar que el tiempo total del movimiento generado es de 3.6 segundos y el simulado es de 10. Para solucionar este problema como lo muestra la figura 106, y como recomendación, es sumar un tiempo más a la trayectoria, y colocar el mismo valor de referencia del tiempo anterior

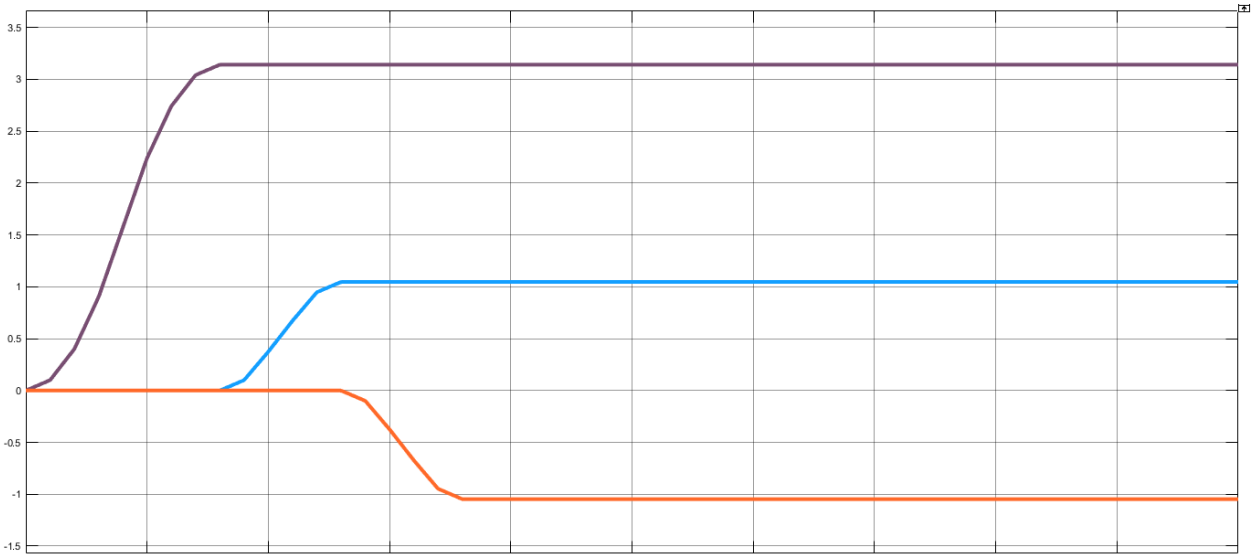


Figure 106. Con ajuste de error de incremento.

CAPÍTULO 7, ACTIVIDADES ESTRÁS

En esta sección se nombra las actividades estrás, realizadas durante la pasantía de investigación, los cuales son la asistencia de unos cursos que se dictan normalmente a estudiantes de pregrado y posgrado pertenecientes a la facultad de ingeniería de la universidad nacional. Como principal objetivo a la asistencia de estas clases, es adquirir conocimiento útil para la ejecución del proyecto. Algo a mencionar es que la asistencia de estos cursos no es calificable. A continuación, se dará una breve explicación de las materias cursadas

7.1 clase de robótica teórica y laboratorio

El curso de robótica se encuentra dividida en dos, una clase es teórica de 2 horas semanales, y la otra clase es práctica también de dos horas semanales. Estas dos materias son dictadas por un profesor diferente. Este curso se dictan temas como; herramientas matemáticas para la localización espacial, cinemática directa del robot, cinemática inversa del robot y matriz jacobiana.

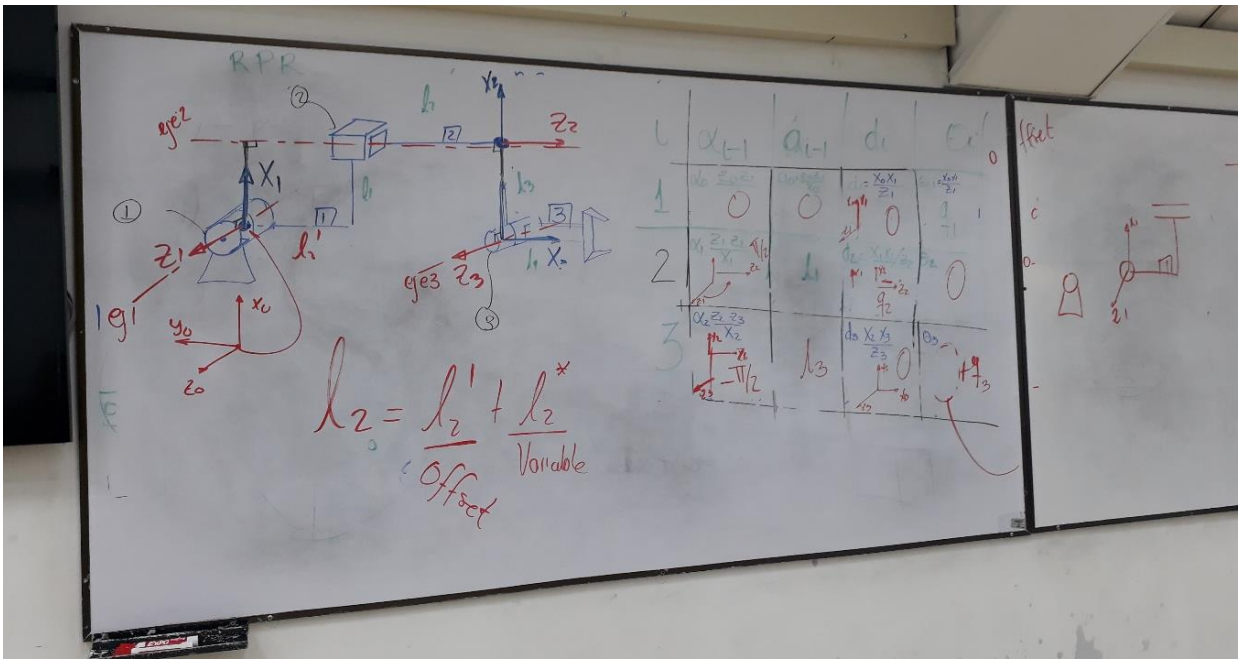


Figure 107. Fotografía de la clase de robótica, tema formulación de denavit-hartenberg.

El curso de laboratorio de robótica, además de poner en practica la teoría aprendida se enseña a utilizar herramientas en el entorno de ROS, como también utilizar los brazos robóticos con el software que ofrece ABB que se llama Robot Studio.

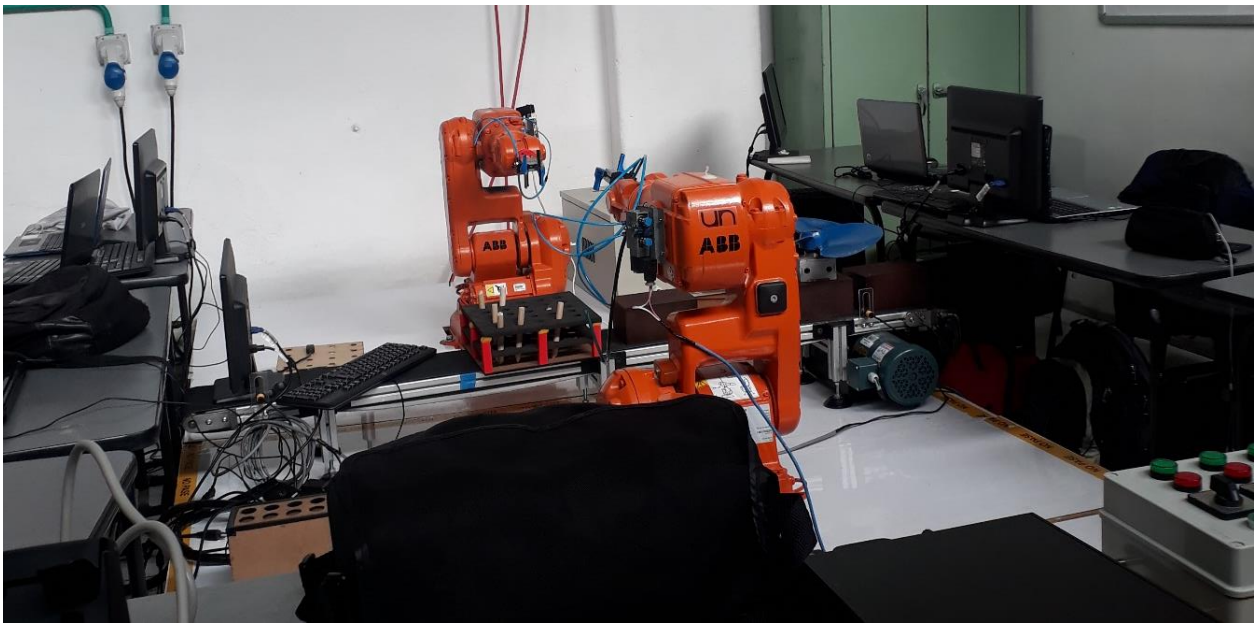


Figure 108. Fotografía en el laboratorio de robótica, brazos robóticos industriales ABB.

7.2 servomecanismo

En el cuso de servo mecanismo es una materia teórica, pero dispone de un proyecto final de aplicación de los temas aprendidos durante el curso. Cuenta con una intensidad de horaria de 4 horas semanales, repartías en dos clases a la semana de dos horas. El curso se enfoca a, tipos de mecanismo, tipos de actuadores, modelado de sistemas y control de sistemas mecatrónicos.

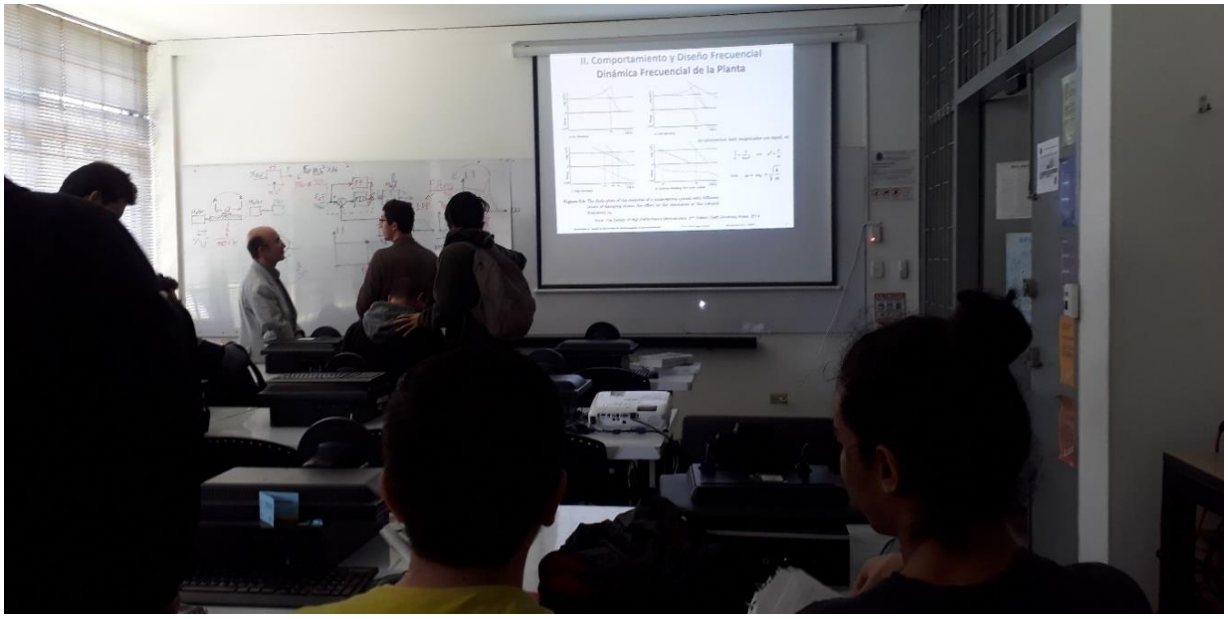


Figure 109. Clase de servomecanismos, tema control de movimiento de sistemas mecatrónicos.

7.3 visión de maquina

Este curso es teórico, cuenta con una intensidad horaria de 4 horas señales repartidas en dos clases. Está enfocado al tratamiento de imágenes, y se enseñan métodos de detección de características, filtros, detectores y descriptores y diversos algoritmos de identificación.

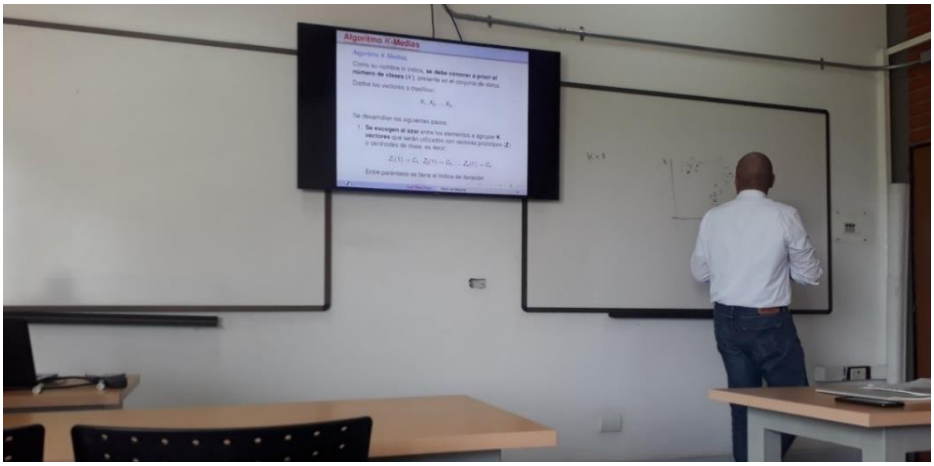


Figure 110. Clase de visión de máquinas.

7.4 Automatización de procesos de manufactura

Este curso cuenta con una intensidad horaria de 4 horas semanales, dividida en dos clases. Está enfocado a temas relacionados con la automatización de procesos industriales, robótica industrial y mechatronics concept designer, en este cuso se utiliza el CAD NX12 y software Robot estudio. Este curso es dictado por diferentes docentes dependiendo el tema a tratar.

Figure 111. Clase de Automatización de procesos de manufactura, Clase de introducción a NX12

CAPÍTULO 8. CONCLUSIONES

Antes que nada, hay que mencionar que Matlab ofrece herramientas muy potentes para el área de sistemas mecatrónicos, que por naturaleza es multidisciplinar. La utilización de herramientas de modelización dinámica como Simscape™, brinda una solución ideal para el diseño, modelación, simulación y optimización de sistemas mecatrónicos complejos como en este caso un brazo robótico, ya que junto con el entorno de Simulink® es una combinación potente, algo que se aprecia durante la etapa de experimentación en dicho entorno, un ejemplo que se presentó durante las pruebas es cuando se intenta llevar las articulaciones a una posición donde físicamente es imposible debido a la colisión con la misma estructura del robot, o cuando el mecanismo de transmisión de movimiento efector final y del antebrazo se llevan a una posición singular las articulaciones se bloquean, como ocurre en la realidad. otros de los ejemplos del buen funcionamiento de este sistema, es la inercia que se genera con movimiento y la posición de las articulaciones simulando muy bien el comportamiento, al igual que en modelado de los actuadores, ofrece un comportamiento muy parecido a la realidad. Esta herramienta permite corroborar la dinámica de nuestro sistema, el controlador para los actuadores, las cinemáticas y las estrategias de control implementas en el proyecto de una forma intuitiva.

Por otro lado, el uso de la cinemática es bastante importante para la implementación del control cinemático, la cinemática directa nos permite saber la posición de cada articulación. la cinemática inversa ayuda a crear una trayectoria de manera fácil, permitiendo determinar el movimiento que debe realizar cada articulación para que el efector final se ubique en punto asignado, claramente mencionando que la solución no es única. Para el caso de este robot son cuatro soluciones, codo arriba y codo abajo más codo arriba y codo abajo, pero con la base girada 180 grados, el algoritmo implementado permite escoger entre codo arriba con un valor de -1 y codo abajo con un valor 1.

Las trayectorias punto a punto y sus tres tipos, como se muestra en los resultados de las pruebas, realiza una interpolación entre los puntos especificados, en el caso del movimiento eje a eje como su nombre lo indica mueve un eje de forma consecutiva, lo cual se puede decir que la única ventaja que tiene, es que presenta un menor consumo de energía en cada instante de tiempo, que puede ser de utilidad para casos donde se esté suministrando baja potencia. En segundo lugar, el movimiento simultaneo de ejes; los actuadores comienzan su movimiento simultáneamente, pero terminan en

un diferente tiempo, esto presenta un inconveniente ya que algunos actuadores se someten a elevadas velocidades y aceleraciones, lo que a la larga significa una reducción de la vida útil de los componentes. En cambio, la trayectoria coordinada; las articulaciones al comenzar y terminar al mismo tiempo, evita estos problemas, lo que conlleva a tener un menor consumo en cada instante de tiempo.

El control de trayectoria continua, el sistema debe hacer que el robot siga la trayectoria lo mejor posible a la especificada, esto se logra asignando el mayor número de puntos posibles dependiendo de la aplicación, pero al asignar mayor número de puntos el consumo de recurso aumentaba, tomando una mayor tiempo en realizar la trayectoria, una opción para no tener un consumo de recursos elevados, citando el código es colocar el menor número puntos posibles al interpolador cubico y aumentar los puntos intermedios ya que ese tipo de cálculo consume menos recursos.

Al realizar pruebas del desempeño del controlador PID para el control de los motores, fue un poco complicado de calibrar las constantes debido al que los motores que se usan son tipo brushless y estos giran a altas rpm, pero se observó que al aumentar la contante proporcional tiende hacer el sistema más rápido, pero aumenta la oscilación, la constante integral ayuda alcanzar la referencia más rápido y la constante integral a disminuir la oscilación.

CAPÍTULO 9. RECOMENDACIONES

Inicialmente en el anteproyecto se fijó como unos de los objetivo la implementación del control dinámico como por ejemplo; control de gravedad, pero debido a que los controladores de movimiento no tienen acceso al control del torque externamente como se menciona anteriormente, no fue posible aplicar esa estrategia de control, algo que es muy recomendable para un buen funcionamiento del brazo robótico frente perturbaciones y también para ilustrar el funcionamiento de este tipo de control, como recomendación seria diseñar o adquirir controlador que para esta aplicación pude ser uno de bajo costo, donde se tenga acceso para realizar control por corriente y así controlar el par.

Respecto al control de los actuadores, para tener una mejor respuesta de funcionamiento, es aconsejable implementar un control PID adaptivo mediante lógica difusa para que el sistema se vaya auto sintonizando cuando lo requiera.

A medida que se desarrolle el proyecto es aconsejable que se vaya documentando por día todo lo experimentado, tanto problemas y soluciones, ya que esto es de gran ayuda a medida que se va avanzando el proyecto, ya que dicha información puede ser mucha utilidad para solucionar problemas. Si es posible, es recomendable si se cuenta con un tiempo reducido que se vaya redactando el trabajo escrito, ya que esto ayuda no tener exceso de trabajo en las etapas finales del proyecto y permite tener más organizadas las ideas. Como también planeamiento y un tiempo de acción para solucionar problemas, como en este caso, no se mecaniza la estructura del robot, por motivos de que la universidad no proporciona los recursos a tiempo, así poder optar por otros medios de forma anticipada para conseguir el objetivo.

REFERENCIAS

Acosta Sánchez, L., & Sigut Saavedra, M. (2005). Matemáticas y robótica. *SCTM05, 3*, 167-182. Obtenido de https://dialnet.unirioja.es/servlet/articulo?codigo=2949940

Arduino®. (s.f.). *Arduino*. Obtenido de https://store.arduino.cc/usa/mega-2560-r3

Arian, A., Danaei, B., Abdi, H., & Nahavandi, S. (November de 2017). Kinematic and dynamic analysis of the Gantry-Tau, a 3-DoF translational parallel manipulator. *Applied Mathematical Modelling, 51*, 217-231. doi:https://doi.org/10.1016/j.apm.2017.06.012

Barrientos Cruz, A., Balaguer, C. B., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robótia* (2 ed.). Madrid: McGraw-Hill. Recuperado el 26 de 03 de 2019, de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Barrientos Cruz, A., Balaguer, C. B., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robotica.* Madrid, España, España: McGraw-Hill. Obtenido de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Barrientos Cruz, A., Balaguer, C. B., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robotica* (2 ed.). Madrid, España: McGraw-Hill. Obtenido de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Barrientos Cruz, A., Balaguer, C. B., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robótica* (2 ed.). Madrid, España: McGraw-Hill. Obtenido de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Barrientos Cruz, A., Balaguer, C. B., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robótica.* Madrid, España: McGraw-Hill. Obtenido de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Barrientos Cruz, A., Balaguer, C. B., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robótica* (2 ed.). Madrid, España: McGraw-Hill. Obtenido de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Barrientos Cruz, A., Balaguer, C. B., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robótica* (2 ed.). Madrid, España: McGraw-Hill. Obtenido de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Barrientos Cruz, A., Balaguer, C., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robótia* (2 ed.). Madrid, España: McGraw-Hill. Recuperado el 26 de 03 de 2019, de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Barrientos Cruz, A., Balaguer, C., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robotica.* Madrid, España: McGraw-Hill. Recuperado el 25 de 03 de 2019, de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Barrientos Cruz, A., Balaguer, C., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robótica* (2 ed.). Madrid: McGraw-Hill. doi:9788448156367

Barrientos Cruz, A., Balaguer, C., Bala, C., Peñin, L. F., & Aracil, R. (2007). *Fundamentos de robótica.* Madrid, España: McGraw-Hill. Obtenido de http://www.ingebook.com/ib/NPcd/IB_BooksVis?cod_primaria=1000187&codigo_libro=4101

Baturone, A. O. (2005). *Robótica, manipuladores y robots móviles.* (Marcombo, Ed.) Marcombo. doi:8426713130

Beauregard:, B. (15 de abril de 2015). *http://brettbeauregard.com.* Obtenido de http://brettbeauregard.com/blog/wp-content/uploads/2012/07/Gu%C3%ADa-de-uso-PID-para-Arduino.pdf

dobot®. (s.f.). *dobot*. Obtenido de https://www.dobot.cc/dobot-m1/product-overview.html

Duro López, M., & Villarejo Mañas, J. A. (25 de 09 de 2017). Modelado y Simulación del Funcionamiento y Control de un Motor BLDC. Cartagena, Colombia. Obtenido de http://repositorio.upct.es/xmlui/bitstream/handle/10317/6475/tfg-dur-mod.pdf?sequence=1&isAllowed=y

Espinosa Zapata, F., Mazo, M., López Guillén, E., & Ureña, J. (1998). Generación de trayectorias y detección de obstáculos mediante splines en el guiado de robots móviles. *Informacion Tecnologica 1998, 9*(6), 125-134. Obtenido de https://books.google.com.co/books?hl=es&lr=&id=jqUlENrmQ6YC&oi=fnd&pg=PA125&dq=Generaci%C3%B3n+de+trayectorias+cartesianas&ots=KnNDeeZKms&sig=BAIYYge1JHWf7b3FbbQlXeDcnF4#v=onepage&q=Generaci%C3%B3n%20de%20trayectorias%20cartesianas&f=false

Faulhaber®. (2015). *Instruction Manual FAULHABER Motion Manager 5.4.* Alemania . Obtenido de https://www.faulhaber.com/en/support/faulhaber-motion-manager/

Faulhaber®. (s.f.). *faulhaber.com*. Obtenido de https://www.faulhaber.com/en/support/technical-support/encoder/documentation-for-encoders/

Faulhaber®. (s.f.). *Manual Motion Controller for Brushless DC-Servomotors* (1 ed.). Alemania. Obtenido de https://www.faulhaber.com/en/support/technical-support/drive-electronics/documentation-for-drive-electronics/

Flórez Vergara, D. É., Castro Riveros, F. C., & Castillo Estepa , R. A. (06 de 2017). planeación y ejecución de trayectorias en un robot delta. *Scientia et Technica, 22*, 183-192. doi:ISSN 0122-1701

Ghariblu, H. (June de 2015). A new mobile ball robot-dynamic modeling and simulation. *Applied Mathematical Modelling, 39*, 3103-3115. doi:https://doi.org/10.1016/j.apm.2014.11.020

Hassan, A., & Abomoharam, M. (August de 2017). Modeling and design optimization of a robot gripper mechanism. *Robotics and Computer-Integrated Manufacturing, 46*, 94-103. doi:https://doi.org/10.1016/j.rcim.2016.12.012

Holly A, Y., & Jill L, D. (2002). A taxonomy for human-robot interaction. *AAAI Technical Report*, 111-119.

Izaguirre, E., Hernández, L., Rubio, E., Prieto, P. J., & Hernández, A. (October-December de 2011). Control Desacoplado de Plataforma Neumática de 3-GDL utilizada como Simulador de Movimiento. *Revista Iberoamericana de Automática e Informática Industrial RIAI, 8*(4), 345-356. doi:https://doi.org/10.1016/j.riai.2011.09.003

Mathworks®. (s.f.). *mathworks.com*. (Mathworks, Editor) Obtenido de https://la.mathworks.com/products/simscape.html

Mathworks®. (s.f.). *mathworks.com.* Obtenido de https://la.mathworks.com/hardware-support/arduino-simulink.html

Mathworks®. (s.f.). *mathworks.com.* Obtenido de https://la.mathworks.com/hardware-support/arduino-matlab.html?s_tid=AO_HS_info

Merckaert, K., De Beir, A., Adriaens, N., El Makrini, I., Van Ham, R., & Vanderborgh, B. (10 de 2018). Independent load carrying and measurement manipulator robot arm for improved payload to mass ratio. *Robotics and Computer-Integrated Manufacturing, 53*, 135-140. doi:https://doi.org/10.1016/j.rcim.2018.04.001

Miguel, M., Blanes, F., Benet, G., Simó, J., & Perez, P. (27-30 de June de 2005). Distributed real time architecture for small biped robot YABIRO. *2005 International Symposium on Computational Intelligence in Robotics and Automation*. doi:10.1109/CIRA.2005.1554351

Milanés Hermosilla, D., & Castilla Pérez, A. (September-December de 2016). Generación de trayectorias para el brazo robótico (ArmX). *EAC, 37*(3). Obtenido de http://scielo.sld.cu/scielo.php?pid=S1815-59282016000300006&script=sci_arttext&tlng=pt

Ning, T., Mohan, R. E., & Akiko, A. (September de 2016). Toward a framework for robot-inclusive environments. *Automation in Construction, 69*, 68-78. doi:https://doi.org/10.1016/j.autcon.2016.06.001

Ollero Baturone, A. (2005). *Robótica, manipuladores y robots móviles.* (Marcombo, Ed.) Marcombo. doi:8426713130

Özgü, E., & Mezouar, Y. (March de 2016). Kinematic modeling and control of a robot arm using unit dual quaternions. *Robotics and Autonomous Systems, 77*, 33-73. doi:https://doi.org/10.1016/j.robot.2015.12.005

Papageorgiou, D., Kastritsi, T., & Doulgeri, Z. (febrero de 2020). A passive robot controller aiding human coaching for kinematic behavior modifications. *Robotics and Computer-Integrated Manufacturing, 61*, 101-824. doi:https://doi.org/10.1016/j.rcim.2019.101824

Renteria , A., & Rivas, M. (2000). *Robotica Industrial Fundamentos y Aplicaciones* (1 ed.). (A. G. Brage, Ed.) Madrid, España: McGRAW W-HILL. doi:978-84-481-2819-7

Rentería, A., & Rivas, M. (200). *Robotica Industrial Fundamentos y Aplicaciones.* (A. G. Brage, Ed.) Madrid, España: McGRAW W-HILL. doi:978-84-481-2819-7

Restrepo, J., Villegas, J., Arias, A., Sergio, S., & Madrigal, C. (November de 2012). Trajectory generation for a robotic in a robocup test scenery using Kalman filter and B-spline curves. *2012 XVII Symposium of Image, Signal Processing, and Artificial Vision (STSIVA)*. doi:10.1109/STSIVA.2012.6340566

Reyes Cortes, F. (2011). *Robotica Control de Robots Manipuladores* (1 ed.). México: Alfaomega. doi:978-607-707-190-7

Reyes Cortes, F. (2011). *Robotica Control de Robots Manipuladores* (1 ed.). México: Alfaomega. doi:978-607-707-190-7

Reyes Cortes, F. (2011). *Robotica Control de Robots Manipuladores* (1 ed.). México: Alfaomega. doi:978-607-707-190-7

Robóticos, P. (s.f.). *proyectosroboticos.com.* Obtenido de https://sites.google.com/site/proyectosroboticos/control-de-motores/control-pid-mejorado

RUÍZ OLAYA, A. F. (Octubre de 2008). Sistema Robótico Multimodal para Análisis y Estudios en Biomecánica, Movimiento Humano y Control Neuromotor. Madrid, España: UNIVERSIDAD CARLOS III DE MADRID. Obtenido de https://s3.amazonaws.com/academia.edu.documents/42564687/EnferQuirurgic.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1555659007&Signature=mdlkr

cwpKxykRTASH9rmz9hJ1SM%3D&response-content-disposition=inline%3B%20filename%3DENFERMERIA_QUIRUJICA.pdf

Sciavicco, L., Siciliano, B., & Villani, L. (September de 1994). On dynamic modelling of gear-driven rigid robot manipulators. *IFAC Proceedings Volumes, 27*, 543-549. doi:https://doi.org/10.1016/S1474-6670(17)47364-2

Shafei, A., & Shafei, H. (August de 2018). Dynamic modeling of planar closed-chain robotic manipulators in flight and impact phases. *Mechanism and Machine Theory, 126*, 141-154. doi:https://doi.org/10.1016/j.mechmachtheory.2018.03.007

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). *Robotics: Modelling, Planning and Control*. Londón: Springer. doi:10.1007/978-1-84628-642-1

Xiong, G., Din, Y., & Zhu, L. (2019). Stiffness-based pose optimization of an industrial robot for five-axis milling. *Robotics and Computer-Integrated Manufacturing, 55*, 19-28. doi:https://doi.org/10.1016/j.rcim.2018.07.001

Y. Nof, S. (March 1999). *Handbook of Industrial Robotics* (2 ed.). Canada: JOHN WILEY & SONS, INC. doi:978-0-471-17783-8

Yanco, H., & Drury, J. (7 de 3 de 2005). Classifying human-robot interaction: an updated taxonomy. *2004 IEEE International Conference on Systems*. doi:10.1109/ICSMC.2004.1400763

Zhang, H., Jin, H., Liu, Z., Liu, Y., Zhu, Y., & Zhao, J. (2019). Real-time Kinematic Control for Redundant Manipulators in a Time-varying Environment: Multiple-dynamic Obstacle Avoidance and Fast Tracking of a Moving Object. *IEEE Transactions on Industrial Informatics*. doi:10.1109/TII.2019.2917392

Zhou, Y., Luo, J., & Wang, M. (July de 2019). Dynamic coupling analysis of multi-arm space robot. *Acta Astronautica, 160*. doi:https://doi.org/10.1016/j.actaastro.2019.02.017

ANEXOS

Anexo A. Código control de posición y velocidad en arduino

```
const byte   encM1_A = 2;          // Entrada de la señal A del encoder I2-64.
const byte   encM1_B = 3;          // Entrada de la señal B del encoder I2-64.
const byte   encM2_A = 18;         // Entrada de la señal A del encoder I2-64.
const byte   encM2_B = 19;         // Entrada de la señal B del encoder I2-64.
const byte   encM3_A = 20;         // Entrada de la señal A del encoder I2-64.
const byte   encM3_B = 21;         // Entrada de la señal B del encoder I2-64.
const byte   PWM_M1 = 4;           // Salida PWM a DAC.
const byte   PWM_M2 = 5;           // Salida PWM a DAC.
const byte   PWM_M3 = 6;           // Salida PWM a DAC.
const byte   fault_pin_M1 = 7;     // Direccion de giro fault pin MCBL 2805.
const byte   fault_pin_M2 = 8;     // Direccion de giro fault pin MCBL 2805.
const byte   fault_pin_M3 = 9;     // Direccion de giro fault pin MCBL 2805.
```

- En segundo lugar, esta parte corres ponte a las variables utilizadas para el PID, específicamente para el PID de un solo motor, para cada control debe utilizar sus propias variables.

```
unsigned long lastTime_1 = 0, SampleTime_1= 0;    // Variables de tiempo discreto.
double       Input_1= 0.0, Setpoint_1= 0.0;       //   "   de posición del motor y posición a la que queremos llevar el motor (posición designada).
double       ITerm_1= 0.0, dInput_1= 0.0, lastInput_1= 0.0;   //   "   de error integral, error derivativo y posición anterior del motor
double    kp_1= 0.0, ki_1= 0.0, kd_1= 0.0;   // Constantes: proporcional, integral y derivativa.
Doublé  outMin_1= 0.0, outMax_1= 0.0;  // Límites para no sobrepasar la resolución del PWM.
double       error_1= 0.0;    // Desviación o error entre la posición real del motor y la posición designada.
```

- Las siguientes variables se utilizan para guardar valores que se generan en el programa

```
volatile long contador_1 = 0;          // En esta variable se guardará los pulsos del encoder y que interpreremos como distancia (o ángulo si ese fuese el caso).
volatile long contador_2 = 0;          // En esta variable se guardará los pulsos del encoder y que interpreremos como distancia (o ángulo si ese fuese el caso).
volatile long contador_3 = 0;          // En esta variable se guardará los pulsos del encoder y que interpreremos como distancia (o ángulo si ese fuese el caso).
byte         pwm_1 = 0;       // Es el PWM, se transformará en voltaje real en las bobinas de los motores.
byte         pwm_2= 0;    // Es el PWM, se transformará en voltaje real en las bobinas de los motores.
```

```
byte        pwm_3 = 0;     // Es el PWM, se transformará en voltaje real en las bobinas de los motores.
byte        cmd = 0;     // Un byte que utilizamos para la comunicación serie. (cmd=comando.)
```

- En la siguiente parte se inician los puertos y variables, interrupciones

```
void setup(void)     // Inicializamos todo las variables que sean necesarias, configuramos los pines de entrada/salida y el terminal serie.
{
 Serial.begin(115200);                 // Configura la velocidad en baudios del terminal serie. Hoy en día "115200" es soportada por la gran mayoría de computadores.
 Serial2.begin(115200);
 Serial3.begin(115200);
 pinMode(PWM_M1, OUTPUT);                 // Declara las dos salidas PWM para el control del motor (pin 4).
  pinMode(PWM_M2, OUTPUT);             //       "          "          "          (pin 5).
  pinMode(PWM_M3, OUTPUT);             //       "          "          "          (pin 6).
  pinMode(fault_pin_M1, OUTPUT);       //       "          "          "          (pin 7).
  pinMode(fault_pin_M2, OUTPUT);       //       "          "          "          (pin 8).
  pinMode(fault_pin_M3, OUTPUT);       //       "          "          "          (pin 9).
  digitalWrite(PWM_M1, LOW);           // salidas se inicializan a cero.
  digitalWrite(PWM_M2, LOW);
  digitalWrite(PWM_M3, LOW);
  digitalWrite(fault_pin_M1, LOW);
  digitalWrite(fault_pin_M2, LOW);
  digitalWrite(fault_pin_M3, LOW);
  TCCR0B = TCCR0B & B11111000 | 1;    // Configuración de la frecuencia del PWM para los pines 5 y 6. https://arduino-info.wikispaces.com/Arduino-PWM-Frequency
// Podemos variar la frecuencia del PWM con un número de 1 (32KHz) hasta 7 (32Hz). El número que pongamos es un divisor de frecuencia. Min.=7, Max.=1. Está a la máxima frecuencia y es como mejor resultado me ha dado y además es silencioso.
  attachInterrupt(digitalPinToInterrupt(encM1_A), encoder_1, CHANGE); // En cualquier flanco ascendente o descendente
  attachInterrupt(digitalPinToInterrupt(encM1_B), encoder_1, CHANGE); // en los pines 2 y 3 actúa la interrupción.M1
  attachInterrupt(digitalPinToInterrupt(encM2_A), encoder_2, CHANGE); // En cualquier flanco ascendente o descendente
  attachInterrupt(digitalPinToInterrupt(encM2_B), encoder_2, CHANGE); // en los pines 2 y 3 actúa la interrupción.
  attachInterrupt(digitalPinToInterrupt(encM3_A), encoder_3, CHANGE); // En cualquier flanco ascendente o descendente
```

```
 attachInterrupt(digitalPinToInterrupt(encM3_B), encoder_3, CHANGE); // en los pines 2 y 3 actúa la interrupción.
 // Acotación máxima y mínima; corresponde a Max.: 0=0V hasta 255=5V (PWMA), y Min.: 0=0V hasta -255=5V (PWMB). El PWM se convertirá a la salida en un valor absoluto, nunca negativo.
outMax_1 =  50.0;                 // Límite máximo del controlador PID.
outMin_1 = -outMax_1;             // Límite mínimo del controlador PID.
outMax_2 =  50.0;                 // Límite máximo del controlador PID.
outMin_2 = -outMax_2;             // Límite mínimo del controlador PID.
outMax_3 =  50.0;                 // Límite máximo del controlador PID.
outMin_3 = -outMax_3;             // Límite mínimo del controlador PID.
SampleTime_1 = 50;                // Se le asigna el tiempo de muestreo en milisegundos.
SampleTime_2 = 50; SampleTime_3 = 50;
kp_1 = 1.0;                       // Constantes PID iniciales. Los valores son los adecuados para un encoder de 334 ppr (con un motor de 12V),
ki_1 = 0.05;                      // pero como el lector de encoder está diseñado como x4, entonces equivale a uno de 1336 ppr. (ppr = pulsos por revolución.)
kd_1 = 25.0;
kp_2 = 1.0;
ki_2 = 0.05;
kd_2 = 25.0;
kp_3 = 1.0;
ki_3 = 0.05;
kd_3 = 25.0;
SetTunings_1(kp_1, ki_1, kd_1);   // Llama a la función de sintonización y le envía los valores que hemos cargado anteriormente.
SetTunings_2(kp_2, ki_2, kd_2);   // Llama a la función de sintonización y le envía los valores que hemos cargado anteriormente.
SetTunings_3(kp_3, ki_3, kd_3);   // Llama a la función de sintonización y le envía los valores que hemos cargado anteriormente.
Setpoint_1 = (double)contador_1;  // Para evitar que haga cosas extrañas al ponerse en marcha o después de resetear, igualamos los dos valores para que comience estando quieto el motor.
Setpoint_2 = (double)contador_2;
Setpoint_3 = (double)contador_3;
imprimir(3);                      // Muestra las constantes de sintonización, el tiempo de muestreo y la posición por el terminal serie.
}
```

- En el loop es donde se llaman las funciones, se leen los valores de entrada, se escriben los valores de salida.

```
void loop(void) {
double Out_1 = Compute_1();              // Llama a la función "Compute_1()" para calcular la desviación y el resultado lo carga en la variable 'Out_1'.
double Out_2 = Compute_2();              // Llama a la función "Compute_1()" para calcular la desviación y el resultado lo carga en la variable 'Out_1'.
double Out_3 = Compute_3();              // Llama a la función "Compute_1()" para calcular la desviación y el resultado lo carga en la variable 'Out_1'.
//  ************************ Control del Motor 1 ************************
if (error_1 == 0.0){        // Cuando está en el punto designado, parar el motor.
digitalWrite(PWM_M1, LOW);          // Pone a 0 los dos pines del puente en H.
digitalWrite(ledok, HIGH);      // Se enciende el led (pin 13) porque ya está en la posición designada.
}
else{  // De no ser igual, significa que el motor ha de girar en un sentido o al contrario; esto lo determina el signo que contiene "Out".
pwm_1 = abs(Out_1);                 // Transfiere a la variable pwm el valor absoluto de Out.
if (Out_1 > 0.0) {        // Gira el motor en un sentido con el PWM correspondiente a su posición.
digitalWrite(fault_pin_M1, HIGH);      // Manda valor bajo, para giro horario
analogWrite(PWM_M1, pwm_1);          // señal PWM al DAC.
}
else {      // Gira el motor en sentido contrario con el PWM correspondiente a su posición.
digitalWrite(fault_pin_M1, LOW);    // Manda valor alto, para giro antihorario
analogWrite(PWM_M1, pwm_1);          // señal PWM al DAC.
} }
// ************************ Control del Motor 2**************************
if (error_2 == 0.0)                 // Cuando está en el punto designado, parar el motor.
  {
digitalWrite(PWM_M2, LOW);          // Pone a 0 los dos pines del puente en H.
digitalWrite(ledok, HIGH);          // Se enciende el led (pin 13) porque ya está en la posición designada.
}
else {                          // De no ser igual, significa que el motor ha de girar en un sentido o al contrario; esto lo determina el signo que contiene "Out".
pwm_2 = abs(Out_2);                 // Transfiere a la variable pwm el valor absoluto de Out.
if (Out_2 > 0.0) {        // Gira el motor en un sentido con el PWM correspondiente a su posición.
digitalWrite(fault_pin_M2, HIGH);    // Manda valor bajo, para giro horario
analogWrite(PWM_M2, pwm_2);          // señal PWM al DAC.
}
```

```
else {                        // Gira el motor en sentido contrario con el PWM correspondiente a su posición.
digitalWrite(fault_pin_M2, LOW);    // Manda valor alto, para giro antihorario
analogWrite(PWM_M2, pwm_2);          // señal PWM al DAC.
 } }
  //*************************Control del Motor 3**************************
if (error_3 == 0.0) {               // Cuando está en el punto designado, parar el motor.
digitalWrite(PWM_M3, LOW);          // Pone a 0 los dos pines del puente en H.
digitalWrite(ledok, HIGH);          // Se enciende el led (pin 13) porque ya está en la posición designada.
}
else                                // De no ser igual, significa que el motor ha de girar en un sentido o al contrario; esto lo determina el signo que contiene "Out".
{
pwm_3 = abs(Out_3);                 // Transfiere a la variable pwm el valor absoluto de Out.
if (Out_3 > 0.0)                    // Gira el motor en un sentido con el PWM correspondiente a su posición.
{
digitalWrite(fault_pin_M3, HIGH);    // Manda valor bajo, para giro horario
analogWrite(PWM_M3, pwm_3);          // señal PWM al DAC.
}
else {                        // Gira el motor en sentido contrario con el PWM correspondiente a su posición.
digitalWrite(fault_pin_M3, LOW);    // Manda valor alto, para giro antihorario
analogWrite(PWM_M3, pwm_3);          // señal PWM al DAC.
 } }
  // Recepción de datos para posicionar el motor, o modificar las constantes PID, o el tiempo de muestreo. Admite posiciones relativas y absolutas.
if (Serial.available() > 0) {         // Comprueba si ha recibido algún dato por el terminal serie.
cmd = 0;                     // Por seguridad "limpiamos" cmd.
cmd = Serial.read();               // "cmd" guarda el byte recibido.
if (cmd > 31) {
byte flags = 0;              // Borramos la bandera que decide lo que hay que imprimir.
if (cmd >  'Z') cmd -= 32;         // Si una letra entra en minúscula la covierte en mayúscula.
if (cmd == '0') { Setpoint_1 = 0.0;     flags = 2; } // Ir a Inicio.
if (cmd == '0') { Setpoint_2 = 0.0;     flags = 2; } // Ir a Inicio.
if (cmd == '0') { Setpoint_3 = 0.0;     flags = 2; } // Ir a Inicio.
f (cmd == '0') { outMax_1 = outMax_1;      flags = 2; }          // Velocidad por defecto.
if (cmd == '0') { outMax_2 = outMax_2;     flags = 2; }          // Velocidad por defecto.
if (cmd == '0') { outMax_3 = outMax_3;     flags = 2; }          // Velocidad por defecto.
// Decodificador para modificar las constantes PID.
```

```
switch(cmd) {                                                    // Si ponemos en el terminal serie, por ejemplo "p2.5 i0.5 d40" y pulsas enter  tomará esos valores y los cargará en kp, ki y kd.
// También se puede poner individualmente, por ejemplo "p5.5", sólo cambiará el parámetro kp, los mismo si son de dos en dos.
case 'P': kp_1  = Serial.parseFloat(); SetTunings_1(kp_1, ki_1, kd_1); flags = 1; break; // Carga las constantes y presenta en el terminal serie los valores de las variables que hayan sido modificadas.
case 'I': ki_1  = Serial.parseFloat(); SetTunings_1(kp_1, ki_1, kd_1); flags = 1; break;
case 'D': kd_1  = Serial.parseFloat(); SetTunings_1(kp_1, ki_1, kd_1); flags = 1; break;
case 'T': SampleTime_1 = Serial.parseInt();              flags = 1; break;
case 'G': Setpoint_1   = Serial.parseFloat();                flags = 2; break; // Esta línea permite introducir una posición absoluta. Ex: g13360 (y luego enter) e irá a esa posición.
case 'L': Setpoint_2   = Serial.parseFloat();                flags = 2; break; // Esta línea permite introducir una posición absoluta. Ex: g13360 (y luego enter) e irá a esa posición.
case 'R': Setpoint_3   = Serial.parseFloat();                flags = 2; break; // Esta línea permite introducir una posición absoluta. Ex: g13360 (y luego enter) e irá a esa posición.
case 'B': outMax_1     = Serial.parseFloat();                flags = 2; break; // Esta línea permite introducir una velocidad max. Ex: 0-255 (y luego enter) e irá a esa velocidad.
case 'N': outMax_2     = Serial.parseFloat();                flags = 2; break; // Esta línea permite introducir una velocidad max. Ex: 0-255 (y luego enter) e irá a esa velocidad.
case 'M': outMax_3     = Serial.parseFloat();                flags = 2; break; // Esta línea permite introducir una velocidad max. Ex: 0-255 (y luego enter) e irá a esa velocidad.
case 'K':     flags = 3; break;
}
if (flags == 2) digitalWrite(ledok, LOW); // Cuando entra una posición nueva se apaga el led y no se volverá a encender hasta que el motor llegue a la posición que le hayamos designado.
imprimir(flags);
} } }
```

A continuación, se muestra el algoritmo PID mas el filtro y función de sintonización, solamente se muestra un control PID, cada motor tiene su algoritmo

```
double Compute_1(void) {
unsigned long now_1 = millis();                 // Toma el número total de milisegundos que hay en ese instante.
unsigned long timeChange_1 = (now_1 - lastTime_1);   // Resta el tiempo actual con el último tiempo que se guardó (esto último se hace al final de esta función).
if(timeChange_1 >= SampleTime_1) {              // Si se cumple el tiempo de muestreo entonces calcula la salida.
Input_1  = (double)contador_1;              // Lee el valor del encoder óptico. El valor del contador se incrementa/decrementa a través de las interrupciones externas (pines 2 y 3).
error_1  = (Setpoint_1 - Input_1)  * kp_1;          // Calcula el error proporcional.
dInput_1 = (Input_1 - lastInput_1) * kd_1;          // Calcula el error derivativo.
```

```
// Esta línea permite dos cosas: 1) Suaviza la llegada a la meta. 2) El error integral se auto-ajusta a las circunstancias del motor.
if (dInput_1==0.0) ITerm_1 += (error_1 * ki_1); else ITerm_1 -= (dInput_1 * ki_1); //Delimita el error integral para eliminar el "efecto windup".
if (ITerm_1 > outMax_1) ITerm_1 = outMax_1; else if (ITerm_1 < outMin_1) ITerm_1 = outMin_1;
double Output_1 = error_1 + ITerm_1 - dInput_1;      // Suma todos los errores, es la salida del control PID.
if (Output_1 > outMax_1) Output_1 = outMax_1; else if (Output_1 < outMin_1) Output_1 = outMin_1; // Acota la salida para que el PWM pueda estar entre outMin y outMax.
lastInput_1 = Input_1;                    // Se guarda la posición para convertirla en pasado.
lastTime_1  = now_1;                      // Se guarda el tiempo   para convertirlo en pasado.
return Output_1;                          // Devuelve el valor de salida PID.
}
}
void SetTunings_1(double kp_1, double ki_1, double kd_1) { // A las constantes KI y KD se les incluyen el tiempo de muestreo. De esta manera se realiza el cálculo una sola vez.
double SampleTimeInSec_1 = ((double)SampleTime_1) / 1000.0;
kp_1 = kp_1;
ki_1 = ki_1 * SampleTimeInSec_1;
kd_1 = kd_1 / SampleTimeInSec_1;
}
```

- La siguiente función lee el encoder, cuando se produzca cualquier cambio en el encoder esta parte hará que incremente o decremento el contador, en ese caso se muestra para un encoder, cada encoder tiene que tener su respectiva función.

```
void encoder_1(void)
{
  if(error_1>0)  contador_1++;
  if(error_1<0)  contador_1--;
}
```

```
function P = cin_directa_3_gdl_paralelo(q_hombro,q_brazo,q_antebrazo)
% dimensiones del robot
q=[q_hombro,q_brazo,q_antebrazo];
L1=148;
L2=135;
L3=147;
% PARAMETROS DENAVIT-HARTENBERT
theta_dh=[q(1)+pi/2, q(2), q(3)];
d_dh=[L1, 0, 0];
a_dh=[0, L2, L3];
alpha_dh=[pi/2, 0, 0];
% CALCULO DE LOS MTH CON RESPECTO AL FRAME MOVIL (relativo a los eslabones)
A01=denavit(theta_dh(1),d_dh(1),a_dh(1),alpha_dh(1));
A12=denavit(theta_dh(2),d_dh(2),a_dh(2),alpha_dh(2));
A23=denavit(theta_dh(3),d_dh(3),a_dh(3),alpha_dh(3));
% calcular las matrices de cada uno de los esl con respecto al sis 0
% MTH MOVIL RELATIVO AL FRAME FIJO
A00=eye(4);
A02=A01*A12;     % extremo "2" referenciado al eje fijo XYZ
A03=A02*A23;     % extremo "3" referenciado al eje fijo XYZ
% ---------------------- RESULTADOS -------------------------------------
P = A03((1:3),4);
```

```
% FUNCION DENAVIT HARTENBERG
function dh = denavit(theta,d,a,alpha)
dh = [cos(theta)    -cos(alpha)*sin(theta) sin(alpha)*sin(theta) a*cos(theta)
      sin(theta)    cos(alpha)*cos(theta) -sin(alpha)*cos(theta) a*sin(theta)
                0               sin(alpha)             cos(alpha)           d
                0                        0                      0           1];
```

```
   function dibujar_robot(PARALELO)
   % Dibuja los ejes de coordenadas
   frame(PARALELO.A00,10,'0'); % Matriz, dimención de la linea, No de eje
coordenado
   frame(PARALELO.A01,10,'1');
   frame(PARALELO.A02,10,'2');
   frame(PARALELO.A03,10,'3');
   % Dibuja las lineas
   dibujar_linea( PARALELO.A00(1:3,4), PARALELO.A01(1:3,4), 'k',3 );
   dibujar_linea( PARALELO.A01(1:3,4), PARALELO.A02(1:3,4), 'k',3 );
   dibujar_linea( PARALELO.A02(1:3,4), PARALELO.A03(1:3,4), 'k',3 );
```

```
% Dibuja_linea(p0,p1,color,grosor);
function dibujar_linea(p0,p1,color,grosor)
% vectores que contienen las componentes de los puntos a dibujar
x = [ p0(1), p1(1)]; y = [ p0(2), p1(2)]; z = [ p0(3), p1(3)];
plot3(x,y,z, 'color',color ,'LineWidth',grosor);
```

```
% funcion para dibujar los ejes de coordenadas del robot
% coordenadas articulares
%
% ejemplo:
%
% clear ,clc
% A = [-0.0000    1.0000    0.0000    0.0000
%  -1.0000   -0.0000    0.0000   10.0000
%   0.0000         0    1.0000   10.0000
%        0         0         0    1.000]
% t = 1    %espesor de la lineas
% subindice = '1'
% frame(A,t,subindice)

function fm=frame(A,t,subindice)
%definición de los puntos característicos del sistema coordenado
espaciado=.5; grosor=1.2;
pna=[0;0;0;1]; pnb=[t;0;0;1];  pnc=[0;t;0;1];  pnd=[0;0;t;1];
%Se realiza el cambio de base, según A
p0a=A*pna; p0b=A*pnb; p0c=A*pnc; p0d=A*pnd;
%Dibujar la línea del eje X
dibujar_linea(p0a,p0b,[1, 0, 0],grosor); hold on
%Dibujar la línea del eje Y
dibujar_linea(p0a,p0c,[0, 1, 0],grosor);
%Dibujar la línea del eje Z
dibujar_linea(p0a,p0d,[0, 0, 1],grosor);
text( p0b(1)+espaciado,p0b(2),p0b(3),strcat('X',subindice) );
text( p0c(1),p0c(2)+espaciado,p0c(3),strcat('Y',subindice) );
text( p0d(1),p0d(2),p0d(3)+espaciado,strcat('Z',subindice) );
```

Anexo D Cinemática inversa (método geométrico).

```
function [q1,q2,q3]= Cinematica_inversa_3_gdl_paralelo(codo,px,py,pz)
% dimensiones del robot %mm
% P = [ 0, 0, 0 ];
L1=148;
L2=135;
L3=147;
% Calulo q1
q1=atan2(-px,py);
% calculo q3
r=sqrt(px^2 + py^2);
J=sqrt(r^2 + (pz-L1)^2);
cosq3=(J^2-L2^2-L3^2)/(2*L2*L3);
% en caso de que ocurra un error y se genere un numero imaginario para el sin
% se debe limitar el rango de la funcion cosq3a de (-1 a 1)
if cosq3 > 1 & cosq3 < 1.00001
    cosq3=1;
    disp('condicion1');
end
if cosq3 < -1 & cosq3 > -1.00001
    cosq3=-1;
    disp('condicion2');
end
senq3=codo*sqrt(1-cosq3^2);
q3=atan2(senq3,cosq3);
% calculo q2
alpha=atan2(pz-L1,r);
beta=atan2(L3*senq3,L2+L3*cosq3);
q2=alpha-beta;
```

```
function trayectoria = int_trapezoidal_npuntos(qi,qf,a,vmax,npuntos,ti)
tao=vmax/a;               % Tiempo total
delta_qlmt=vmax*tao;      % Drlta de qlimite
delta_q=abs(qf-qi);         % Drlta de q, Como es distancia colocamos valor
absoluto
s=sign(qf-qi);            % Devuelve el signo de la expresión +1 o -1
% Caso trapezoidal
if delta_qlmt<delta_q
    t2=(delta_q-(tao*vmax))/vmax;
    tmin=t2+2*tao;
end
if delta_qlmt==delta_q
    tmin=2*tao;
end
% Caso triangular
if delta_qlmt>delta_q
    tao=sqrt(delta_q/a);
    tmin=2*tao;
    vmax=a*tao;
end
%
tf=tmin+ti;
vec_t=linspace(ti,tf,npuntos);      % Hallamos npuntos desde ti hastas tf,
tao_b=s*(qf-qi)/vmax+vmax/a;
for i=1:npuntos
    %   Arranque
    if vec_t(i)<=tao+ti
        vec_q(i)=qi+s*(a/2)*(vec_t(i)-ti)^2;
        vec_qp(i)=s*a*(vec_t(i)-ti);
        vec_qpp(i)=s*a;
    end
    %   Mantenimiento
    if vec_t(i)>tao+ti && vec_t(i)<=tf-tao
        vec_q(i)=qi-s*(vmax^2)/(2*a)+s*vmax*(vec_t(i)-ti);
        vec_qp(i)=s*vmax;
        vec_qpp(i)=0;
    end
    %   Frenado
    if vec_t(i)>tf-tao && vec_t(i)<=tf
        vec_q(i)=qf+s*(((-a*tao_b^2)/2)+(a*tao_b*(vec_t(i)-ti))-
(a/2)*(vec_t(i)-ti)^2);
        vec_qp(i)=s*(a*tao_b-a*(vec_t(i)-ti));
        vec_qpp(i)=-s*a;
    end
end
% ----------------------------- RESULTADOS ---------------------------------
----
trayectoria.vec_t = vec_t;
trayectoria.vec_q = vec_q;
trayectoria.vec_qp = vec_qp;
trayectoria.vec_qpp = vec_qpp;
end
```

```
function trayectoria = int_trapezoidal_t_npuntos(qi,qf,a,vmax,npuntos,ti,tf)
% vec_q=zeros(1,npuntos); vec_qp=zeros(1,npuntos); vec_qpp=zeros(1,npuntos);
tmin=0;
tao=vmax/a;               % Tiempo total
delta_qlmt=vmax*tao;      % Drlta de qlimite
delta_q=abs(qf-qi);         % Drlta de q, Como es distancia colocamos valor
absoluto
s=sign(qf-qi);            % Devuelve el signo de la expresión +1 o -1
% Caso trapezoidal
if delta_qlmt<delta_q
    t2=(delta_q-(tao*vmax))/vmax;
    tmin=t2+2*tao;
end
if delta_qlmt==delta_q
    tmin=2*tao;
end
% Caso triangular
if delta_qlmt>delta_q
    tao=sqrt(delta_q/a);
    tmin=2*tao;
    vmax=a*tao;
end
vec_t=linspace(ti,tf,npuntos);      % Hallamos npuntos desde ti hastas tf,
% Mandamos un error cuando delta_t<tmin
if (tf-ti)<tmin
    error('??? Error using ==> int_trapezoidal_t_npuntos at 61 se requiere de
más tiempo para realizar el movimiento');
end
if (tf-ti)==tmin
tao_b=s*(qf-qi)/vmax+vmax/a;
for i=1:npuntos
    %   Arranque
    if vec_t(i)<=tao+ti
        vec_q(i)=qi+s*(a/2)*(vec_t(i)-ti)^2;
        vec_qp(i)=s*a*(vec_t(i)-ti);
        vec_qpp(i)=s*a;
    end
    %   Mantenimiento
    if vec_t(i)>tao+ti && vec_t(i)<=tf-tao
        vec_q(i)=qi-s*(vmax^2)/(2*a)+s*vmax*(vec_t(i)-ti);
        vec_qp(i)=s*vmax;
        vec_qpp(i)=0;
    end
    %   Frenado
    if vec_t(i)>tf-tao && vec_t(i)<=tf
        vec_q(i)=qf+s*(((-a*tao_b^2)/2)+(a*tao_b*(vec_t(i)-ti))-
(a/2)*(vec_t(i)-ti)^2);
        vec_qp(i)=s*(a*tao_b-a*(vec_t(i)-ti));
        vec_qpp(i)=-s*a;
    end
end
end
% Caso triangular
if (tf-ti)>tmin
    vp=[1/a,ti-tf,delta_q];
    vp=roots(vp);
    if vp(1)<=vmax
```

```
        vmax=vp(1);
    else
        vmax=vp(2);
    end
    tao=vmax/a;
    tao_b=s*(qf-qi)/vmax+vmax/a;
    for i=1:npuntos
    %   Arranque
    if vec_t(i)<=tao+ti
        vec_q(i)=qi+s*(a/2)*(vec_t(i)-ti)^2;
        vec_qp(i)=s*a*(vec_t(i)-ti);
        vec_qpp(i)=s*a;
    end
    %   Mantenimiento
    if vec_t(i)>tao+ti && vec_t(i)<=tf-tao
        vec_q(i)=real(qi-s*(vmax^2)/(2*a)+s*vmax*(vec_t(i)-ti));
        vec_qp(i)=real(s*vmax);
        vec_qpp(i)=0;
    end
    %   Frenado
    if vec_t(i)>tf-tao && vec_t(i)<=tf
        vec_q(i)=real(qf+s*(((-a*tao_b^2)/2)+(a*tao_b*(vec_t(i)-ti))-
(a/2)*(vec_t(i)-ti)^2));
        vec_qp(i)=real(s*(a*tao_b-a*(vec_t(i)-ti)));
        vec_qpp(i)=-s*a;
    end
end
end
% ---------------------------- RESULTADOS ---------------------------------
----
trayectoria.vec_t = vec_t;
trayectoria.vec_q = vec_q;
trayectoria.vec_qp = vec_qp;
trayectoria.vec_qpp = vec_qpp;
end
```

```
function tmin = int_trapezoidal_tmin(qi, qf, a, vmax)
tmin = 0;
delta_q = abs(qf-qi);
tao = vmax/a;
delta_qlimite = tao*vmax;
% Caso trapezoidal
if delta_q > delta_qlimite
    tmin = (delta_q-tao*vmax+2*tao*vmax)/vmax;
end
if delta_q == delta_qlimite
    tmin = 2*tao;
end
% Caso triangular
if delta_q < delta_qlimite
    tao = sqrt(delta_q/a);
    tmin = 2*tao;
end
end
```

```
% Movimiento_eje_a_eje.m
ti=zeros(1,3); % q_paralelo=zeros(1,3);
% Ejemplo
qi = [0,0,0]; qf = [pi/3,pi/6,-pi/3]; a = [5,5,5]; vmax = [5,5,5]; ti = 0;
% Calculo del tiempo minimo que se puede realizar la trayecoria
for i=1:3
tmin(i) = int_trapezoidal_tmin(qi(i), qf(i), a(i), vmax(i));
end
% Calculo del numero de puntos de la trayectoria
for j=1:3
delta_t= 0.2;
tmina(j) = ceil(tmin(j)/delta_t)*delta_t;   % Redondea el No mas cercano
mayor
npuntos(j) = fix(tmina(j)/delta_t)+1;    % Tomamos el valor entero del No
con decimal y le sumamos 1 para completar el numero de puntos
tf(j) = ti+tmina(j);
end
% Determino el tiempo donde inicia cada articulación
q_paralelo=zeros(npuntos(3),3);
ti(1) = ti;
ti(2) = tf(1);
ti(3) = tf(1)+tf(2);
% Calculo de trayectoria de cada articulación
trayectoria1 =
int_trapezoidal_t_npuntos(qi(1),qf(1),a(1),vmax(1),npuntos(1),ti(1),tf(1));
trayectoria2 =
int_trapezoidal_t_npuntos(qi(2),qf(2),a(2),vmax(2),npuntos(2),ti(2),tf(1)+tf
(2));
trayectoria3 =
int_trapezoidal_t_npuntos(qi(3),qf(3),a(3),vmax(3),npuntos(3),ti(3),tf(1)+tf
(2)+tf(3));

TF_tray_total = tf(1)+tf(2)+tf(3);    % Es el tiempo total de las
trayectorias

% ----------------------------------- Trayectoria eje a eje ---------------
---------------------
    % Articulación 1
    vec_t_aux1 = 0 : delta_t : trayectoria1.vec_t(1)- delta_t;
    vec_t_aux2 = trayectoria1.vec_t(end)+ delta_t : delta_t : TF_tray_total;
    trayectoria_eje_a_eje1.vec_t = [vec_t_aux1, trayectoria1.vec_t,
vec_t_aux2 ];
    long1=length(trayectoria1.vec_q); % Determina el tamaño de la
trayectoria del vector de la articulación
    vec_q_aux1 = zeros(1, length(vec_t_aux1));  % Es para agregar los valor
cero en los tiempo iniciales donde no hay movimiento
    vec_q_aux2 = ones(1, length(vec_t_aux2))*trayectoria1.vec_q(long1);  %
Agrega valores finales una vez alcanzada la referencia
    vec_qp_aux1 = zeros(1, length(vec_t_aux1));
    vec_qp_aux2 = zeros(1, length(vec_t_aux2));
    vec_qpp_aux1 = zeros(1, length(vec_t_aux1));
    vec_qpp_aux2 = zeros(1, length(vec_t_aux2));
```

```
    trayectoria_eje_a_eje1.vec_q = [vec_q_aux1, trayectoria1.vec_q,
vec_q_aux2 ];
    trayectoria_eje_a_eje1.vec_qp = [vec_qp_aux1, trayectoria1.vec_qp,
vec_qp_aux2 ];
    trayectoria_eje_a_eje1.vec_qpp = [vec_qpp_aux1, trayectoria1.vec_qpp,
vec_qpp_aux2 ];
    % Articulación 2
    vec2_t_aux1 = 0 : delta_t : trayectoria2.vec_t(1)- delta_t;
    vec2_t_aux2 = trayectoria2.vec_t(end)+ delta_t : delta_t :
TF_tray_total;
    trayectoria_eje_a_eje2.vec_t = [vec2_t_aux1, trayectoria2.vec_t,
vec2_t_aux2 ];
    long2=length(trayectoria2.vec_q); % Determina el tamaño del vector de la
trayectoria de la articulación
    vec2_q_aux1 = zeros(1, length(vec2_t_aux1));    % Es para agregar los
valor cero en los tiempo iniciales donde no hay movimiento
    vec2_q_aux2 = ones(1, length(vec2_t_aux2))*trayectoria2.vec_q(long2);  %
Agrega valores finales una vez alcanzada la referencia;
    vec2_qp_aux1 = zeros(1, length(vec2_t_aux1));
    vec2_qp_aux2 = zeros(1, length(vec2_t_aux2));
    vec2_qpp_aux1 = zeros(1, length(vec2_t_aux1));
    vec2_qpp_aux2 = zeros(1, length(vec2_t_aux2));
    trayectoria_eje_a_eje2.vec_q = [vec2_q_aux1, trayectoria2.vec_q,
vec2_q_aux2 ];
    trayectoria_eje_a_eje2.vec_qp = [vec2_qp_aux1, trayectoria2.vec_qp,
vec2_qp_aux2 ];
    trayectoria_eje_a_eje2.vec_qpp = [vec2_qpp_aux1, trayectoria2.vec_qpp,
vec2_qpp_aux2 ];
    % Articulación 3
    vec3_t_aux1 = 0 : delta_t : trayectoria3.vec_t(1)- delta_t;
    vec3_t_aux2 = trayectoria3.vec_t(end)+ delta_t : delta_t :
TF_tray_total;
    trayectoria_eje_a_eje3.vec_t = [vec3_t_aux1, trayectoria3.vec_t,
vec3_t_aux2 ];
    long3=length(trayectoria3.vec_q); % Determina el tamaño del vector de la
trayectoria de la articulación
    vec3_q_aux1 = zeros(1, length(vec3_t_aux1));    % Es para agregar los
valor cero en los tiempo iniciales donde no hay movimiento
    vec3_q_aux2 = ones(1, length(vec3_t_aux2))*trayectoria3.vec_q(long3);  %
Agrega valores finales una vez alcanzada la referencia;
    vec3_qp_aux1 = zeros(1, length(vec3_t_aux1));
    vec3_qp_aux2 = zeros(1, length(vec3_t_aux2));
    vec3_qpp_aux1 = zeros(1, length(vec3_t_aux1));
    vec3_qpp_aux2 = zeros(1, length(vec3_t_aux2));
    trayectoria_eje_a_eje3.vec_q = [vec3_q_aux1, trayectoria3.vec_q,
vec3_q_aux2 ];
    trayectoria_eje_a_eje3.vec_qp = [vec3_qp_aux1, trayectoria3.vec_qp,
vec3_qp_aux2 ];
    trayectoria_eje_a_eje3.vec_qpp = [vec3_qpp_aux1, trayectoria3.vec_qpp,
vec3_qpp_aux2 ];
    valor=length(trayectoria_eje_a_eje1.vec_t);
    % LINEAS PARA VIDEO
    f1 = figure(1); set(f1,'Color',[1, 1, 1]);
    writerObj = VideoWriter('Ejes a eje.mp4');
    writerObj.FrameRate = 30;
    open(writerObj);
    nv = 3; % numero de veces que añade cada frame al video
```

```
for i=1:valor
    q_paralelo(i,1:3) = [
trayectoria_eje_a_eje1.vec_q(1,i),trayectoria_eje_a_eje2.vec_q(1,i),trayecto
ria_eje_a_eje3.vec_q(1,i)];
    qp_paralelo(i,1:3) = [
trayectoria_eje_a_eje1.vec_qp(1,i),trayectoria_eje_a_eje2.vec_qp(1,i),trayec
toria_eje_a_eje3.vec_qp(1,i)];
    q = [
trayectoria_eje_a_eje1.vec_q(1,i),trayectoria_eje_a_eje2.vec_q(1,i),trayecto
ria_eje_a_eje3.vec_q(1,i)];
    tiempo_p(i,1:3) = [
trayectoria_eje_a_eje1.vec_t(1,i),trayectoria_eje_a_eje2.vec_t(1,i),trayecto
ria_eje_a_eje3.vec_t(1,i) ];
    PARALELO = cin_directa_3_gdl_paralelo(q);
    % Animación del brazo
    figure(1);
    clf(figure(1))
    dibujar_robot(PARALELO)
    axis([-250 250 -250 250 0 400]); grid on;
    xlabel('x');, ylabel('y');, zlabel('z');
    for nv = 1:nv                                   % LINEAS PARA VIDEO
        frame = getframe(f1);                       % LINEAS PARA VIDEO
        writeVideo(writerObj,frame);                % LINEAS PARA VIDEO
    end
end
% Esta linea evita el error de incremento del ultimo eslabón en simulink
q_paralelo(valor+1,3)=trayectoria3.vec_q(long3);
q_paralelo(valor+1,2)=trayectoria2.vec_q(long2);
q_paralelo(valor+1,1)=trayectoria1.vec_q(long1);
qp_paralelo(valor+1,3)=trayectoria3.vec_qp(long3);
qp_paralelo(valor+1,2)=trayectoria2.vec_qp(long2);
qp_paralelo(valor+1,1)=trayectoria1.vec_qp(long1);
cal3=tiempo_p(valor,3)+tiempo_p(2,3);
cal2=tiempo_p(valor,2)+tiempo_p(2,2);
cal1=tiempo_p(valor,1)+tiempo_p(2,1);
tiempo_p(valor+1,3)=cal3;
tiempo_p(valor+1,2)=cal2;
tiempo_p(valor+1,1)=cal1;
% ---------------------------- VARIABLES ARTICULARES -----------------------
-------
q1_eje_a_eje = timeseries(q_paralelo(:,1), tiempo_p(:,1), 'name',
'Position');
q2_eje_a_eje = timeseries(q_paralelo(:,2), tiempo_p(:,2), 'name',
'Position');
q3_eje_a_eje = timeseries(q_paralelo(:,3), tiempo_p(:,3), 'name',
'Position');
qp1_eje_a_eje = timeseries(qp_paralelo(:,1), tiempo_p(:,1), 'name',
'Velocidad');
qp2_eje_a_eje = timeseries(qp_paralelo(:,2), tiempo_p(:,2), 'name',
'Velocidad');
qp3_eje_a_eje = timeseries(qp_paralelo(:,3), tiempo_p(:,3), 'name',
'Velocidad');
  % ------------------------------ GRAFICAS ------------------------------
----
% POSICIÓN
figure(2)
subplot(4,1,1);
```

```
plot (trayectoria_eje_a_eje1.vec_t, trayectoria_eje_a_eje1.vec_q,'o-');,
title('Posición');
grid on
subplot(4,1,2);
plot (trayectoria_eje_a_eje2.vec_t, trayectoria_eje_a_eje2.vec_q,'o-');,
title('Posición');
grid on
subplot(4,1,3);
plot (trayectoria_eje_a_eje3.vec_t, trayectoria_eje_a_eje3.vec_q,'o-');,
title('Posición');
grid on
% VELOCIDAD
figure(3)
subplot(4,1,1);
plot (trayectoria_eje_a_eje1.vec_t, trayectoria_eje_a_eje1.vec_qp,'o-');,
title('velocidad');
grid on
subplot(4,1,2);
plot (trayectoria_eje_a_eje2.vec_t, trayectoria_eje_a_eje2.vec_qp,'o-');,
title('velocidad');
grid on
subplot(4,1,3);
plot (trayectoria_eje_a_eje3.vec_t, trayectoria_eje_a_eje3.vec_qp,'o-');,
title('velocidad');
grid on
% ACELERACIÓN
figure(4)
subplot(4,1,1);
plot (trayectoria_eje_a_eje1.vec_t, trayectoria_eje_a_eje1.vec_qpp,'o-');,
title('Aceleración');
grid on
subplot(4,1,2);
plot (trayectoria_eje_a_eje2.vec_t, trayectoria_eje_a_eje2.vec_qpp,'o-');,
title('Aceleración');
grid on
subplot(4,1,3);
plot (trayectoria_eje_a_eje3.vec_t, trayectoria_eje_a_eje3.vec_qpp,'o-');,
title('Aceleración');
grid on
```

```
% Trayectoria_ejes_simultaneos.m
% EJEMPLO 1
qi = [0,0,0]; qf = [pi,pi/3,-pi/3]; a = [5,5,5]; vmax = [5,5,5]; ti = 0;
deltat=0.3; % menor es mas presición EJ: 0.1
% Calculo del tiempo minimo de cada articulación
for i=1:3
    tmin(i) = int_trapezoidal_tmin(qi(i),qf(i),a(i),vmax(i));
end
for i=1:3
    tmina(i) = ceil(tmin(i)/deltat)*deltat; % Redondea el No mas cercano
mayor
    npuntos(i) = fix(tmina(i)/deltat)+1;   % Tomamos el valor entero del No
con decimal y le sumamos 1 para completar el numero de puntos
end
npuntos_max=max(npuntos);    % El max de los npuntos
valor1=1; valor2=1; valor3=1; valor4=1;
switch npuntos_max
    case npuntos(1)
        valor1=0;
        case npuntos(2)
        valor2=0;
        case npuntos(3)
        valor3=0;
        case npuntos(4)
        valor4=0;
end
% -------------------- Interpolador trapezoidal --------------------------
% ARTICULACIÓN 1
art1=int_trapezoidal_npuntos(qi(1),qf(1),a(1),vmax(1),npuntos(1),ti);
if valor1 ~= 0
art1.vec_t=[art1.vec_t, (1:npuntos_max-npuntos(1))*deltat+art1.vec_t(end)];
art1.vec_q=[art1.vec_q, ones(1,npuntos_max-
npuntos(1))*art1.vec_q(npuntos(1))];
art1.vec_qp=[art1.vec_qp, zeros(1,npuntos_max-npuntos(1))];
art1.vec_qpp=[art1.vec_qpp, zeros(1,npuntos_max-npuntos(1))];
end
% ARTICULACIÓN 2
art2=int_trapezoidal_npuntos(qi(2),qf(2),a(2),vmax(2),npuntos(2),ti);
if valor2 ~= 0
art2.vec_t=[art2.vec_t, (1:npuntos_max-npuntos(2))*deltat+art2.vec_t(end)];
art2.vec_q=[art2.vec_q, ones(1,npuntos_max-
npuntos(2))*art2.vec_q(npuntos(2))];
art2.vec_qp=[art2.vec_qp, zeros(1,npuntos_max-npuntos(2))];
art2.vec_qpp=[art2.vec_qpp, zeros(1,npuntos_max-npuntos(2))];
end
% ARTICULACIÓN 3
art3=int_trapezoidal_npuntos(qi(3),qf(3),a(3),vmax(3),npuntos(3),ti);
if valor3 ~= 0
art3.vec_t=[art3.vec_t, (1:npuntos_max-npuntos(3))*deltat+art3.vec_t(end)];
art3.vec_q=[art3.vec_q, ones(1,npuntos_max-
npuntos(3))*art3.vec_q(npuntos(3))];
art3.vec_qp=[art3.vec_qp, zeros(1,npuntos_max-npuntos(3))];
art3.vec_qpp=[art3.vec_qpp, zeros(1,npuntos_max-npuntos(3))];
end
```

```
% LINEAS PARA VIDEO
f1 = figure(1); set(f1,'Color',[1, 1, 1]);
writerObj = VideoWriter('Ejes simultaneos.mp4');
writerObj.FrameRate = 30;
open(writerObj);
nv = 3; % numero de veces que añade cada frame al video
for i=1:npuntos_max
    q_matriz_simultaneo(i,1:3) = [
art1.vec_q(1,i),art2.vec_q(1,i),art3.vec_q(1,i)];
    tiempo_simultaneo(i,1:3) = [
art1.vec_t(1,i),art2.vec_t(1,i),art3.vec_t(1,i)];
    q_simultaneo = [ art1.vec_q(1,i),art2.vec_q(1,i),art3.vec_q(1,i)];
    PARALELO = cin_directa_3_gdl_paralelo(q_simultaneo);
    % Animación del brazo
    figure(1);
    clf(figure(1))
    dibujar_robot(PARALELO);
    axis([-250 250 -250 250 0 400]); grid on;
    xlabel('x');, ylabel('y');, zlabel('z');
    for nv = 1:nv                                    % LINEAS PARA VIDEO
        frame = getframe(f1);                        % LINEAS PARA VIDEO
        writeVideo(writerObj,frame);                 % LINEAS PARA VIDEO
    end
end
% Esta linea evita el error de incremento del ultimo eslabón en simulink
q_matriz_simultaneo(npuntos_max+1,3)=art3.vec_q(npuntos(3));
q_matriz_simultaneo(npuntos_max+1,2)=art2.vec_q(npuntos(2));
q_matriz_simultaneo(npuntos_max+1,1)=art1.vec_q(npuntos(1));
l3=tiempo_simultaneo(npuntos_max,3)+tiempo_simultaneo(2,3);
l2=tiempo_simultaneo(npuntos_max,2)+tiempo_simultaneo(2,2);
l1=tiempo_simultaneo(npuntos_max,1)+tiempo_simultaneo(2,1);
tiempo_simultaneo(npuntos_max+1,3)=l3;
tiempo_simultaneo(npuntos_max+1,2)=l2;
tiempo_simultaneo(npuntos_max+1,1)=l1;
% ---------------------------- VARIABLES ARTICULARES -----------------------
-------
q1_simultaneo = timeseries(q_matriz_simultaneo(:,1), tiempo_simultaneo(:,1),
'name', 'Position');
q2_simultaneo = timeseries(q_matriz_simultaneo(:,2), tiempo_simultaneo(:,2),
'name', 'Position');
q3_simultaneo = timeseries(q_matriz_simultaneo(:,3), tiempo_simultaneo(:,3),
'name', 'Position');
% --------------------------------- GRAFICAS -----------------------------
figure(2)
subplot(4,1,1);
plot(art1.vec_t,art1.vec_q,'o-'), title('Posición'), ylabel('q1');   %
Tiempo, Posición
grid on
subplot(4,1,2);
plot(art2.vec_t,art2.vec_q,'o-'), title('Posición'), ylabel('q2');  %
Tiempo, Posición
grid on
subplot(4,1,3);
plot(art3.vec_t,art3.vec_q,'o-'), title('Posición'), ylabel('q3');  %
Tiempo, Posición
grid on
figure(3)
```

```
subplot(4,1,1)
plot(art1.vec_t,art1.vec_qp,'o-'), title('Velocidad'), ylabel('q1');   %
Tiempo, velocidad
grid on
subplot(4,1,2)
plot(art2.vec_t,art2.vec_qp,'o-'), title('Velocidad'), ylabel('q2');   %
Tiempo, velocidad
grid on
subplot(4,1,3)
plot(art3.vec_t,art3.vec_qp,'o-'), title('Velocidad'), ylabel('q3');   %
Tiempo, velocidad
grid on

figure(4)
subplot(4,1,1)
plot(art1.vec_t,art1.vec_qpp,'o-'), title('Aceleración'), ylabel('q1');   %
Tiempo, Aceleración
grid on
subplot(4,1,2)
plot(art2.vec_t,art2.vec_qpp,'o-'), title('Aceleración'), ylabel('q2');  %
Tiempo, Aceleración
grid on
subplot(4,1,3)
plot(art3.vec_t,art3.vec_qpp,'o-'), title('Aceleración'), ylabel('q3');   %
Tiempo, Aceleración
grid on
```

```
% Movimiento_coordinado.m
% EJEMPLO 1
qi = [0,0,0]; qf = [pi/3,pi/3,-pi/3]; a = [5,5,5]; vmax = [5,5,5]; ti = 0;
ti=0; npuntos=10;
% Calculo del tiempo minimo
for i=1:3
    tmin(i) = int_trapezoidal_tmin(qi(i),qf(i),a(i),vmax(i));   % Calculamos
el tmin de cada eslabón
end
tf=max(tmin);   % tf es el maximo de la matriz tmin
% Interpolador trapezoidal
art1=int_trapezoidal_t_npuntos(qi(1),qf(1),a(1),vmax(1),npuntos,ti,tf);
art2=int_trapezoidal_t_npuntos(qi(2),qf(2),a(2),vmax(2),npuntos,ti,tf);
art3=int_trapezoidal_t_npuntos(qi(3),qf(3),a(3),vmax(3),npuntos,ti,tf);
% LINEAS PARA VIDEO
f1 = figure(1); set(f1,'Color',[1, 1, 1]);
writerObj = VideoWriter('Coordinada.mp4');
writerObj.FrameRate = 30;
open(writerObj);
nv = 3; % numero de veces que añade cada frame al video
for i=1:npuntos
    q_matriz_coordinada(i,1:3) = [
art1.vec_q(1,i),art2.vec_q(1,i),art3.vec_q(1,i)];
    tiempo_coordinada(i,1:3) = [
art1.vec_t(1,i),art2.vec_t(1,i),art3.vec_t(1,i)];
    q_coordinada = [ art1.vec_q(1,i),art2.vec_q(1,i),art3.vec_q(1,i)];
    coordinada = cin_directa_3_gdl_paralelo(q_coordinada)
    % Animación del brazo
    figure(1);
    clf(figure(1))
    dibujar_robot(coordinada)
    axis([-250 250 -250 250 0 400]); grid on;
    xlabel('x');, ylabel('y');, zlabel('z');
    rotate3d
    for nv = 1:nv                                          % LINEAS PARA VIDEO
        frame = getframe(f1);                              % LINEAS PARA VIDEO
        writeVideo(writerObj,frame);                       % LINEAS PARA VIDEO
    end
end
% Esta linea evita el error de incremento del ultimo eslabón en simulink
q_matriz_coordinada(npuntos+1,3)=art3.vec_q(npuntos);
q_matriz_coordinada(npuntos+1,2)=art2.vec_q(npuntos);
q_matriz_coordinada(npuntos+1,1)=art1.vec_q(npuntos);
p3=tiempo_coordinada(npuntos,3)+tiempo_coordinada(2,3);
p2=tiempo_coordinada(npuntos,2)+tiempo_coordinada(2,2);
p1=tiempo_coordinada(npuntos,1)+tiempo_coordinada(2,1);
tiempo_coordinada(npuntos+1,3)=p3;
tiempo_coordinada(npuntos+1,2)=p2;
tiempo_coordinada(npuntos+1,1)=p1;
% ---------------------------- VARIABLES ARTICULARES -----------------------
-------
q1_coordinada = timeseries(q_matriz_coordinada(:,1), tiempo_coordinada(:,1),
'name', 'Position');
```

```
q2_coordinada = timeseries(q_matriz_coordinada(:,2), tiempo_coordinada(:,2),
'name', 'Position');
q3_coordinada = timeseries(q_matriz_coordinada(:,3), tiempo_coordinada(:,3),
'name', 'Position');
% -------------------------------- GRAFICAS -----------------------------
figure(2)
subplot(4,1,1);
plot(art1.vec_t,art1.vec_q,'o-'), title('Posición'), ylabel('q1');   %
Tiempo, Posición
grid on
subplot(4,1,2);
plot(art2.vec_t,art2.vec_q,'o-'), title('Posición'), ylabel('q2');  %
Tiempo, Posición
grid on
subplot(4,1,3);
plot(art3.vec_t,art3.vec_q,'o-'), title('Posición'), ylabel('q3');  %
Tiempo, Posición
grid on

figure(3)
subplot(4,1,1)
plot(art1.vec_t,art1.vec_qp,'o-'), title('Velocidad'), ylabel('q1');   %
Tiempo, velocidad
grid on
subplot(4,1,2)
plot(art2.vec_t,art2.vec_qp,'o-'), title('Velocidad'), ylabel('q2');   %
Tiempo, velocidad
grid on
subplot(4,1,3)
plot(art3.vec_t,art3.vec_qp,'o-'), title('Velocidad'), ylabel('q3');   %
Tiempo, velocidad
grid on

figure(4)
subplot(4,1,1)
plot(art1.vec_t,art1.vec_qpp,'o-'), title('Aceleración'), ylabel('q1');   %
Tiempo, Aceleración
grid on
subplot(4,1,2)
plot(art2.vec_t,art2.vec_qpp,'o-'), title('Aceleración'), ylabel('q2');  %
Tiempo, Aceleración
grid on
subplot(4,1,3)
plot(art3.vec_t,art3.vec_qpp,'o-'), title('Aceleración'), ylabel('q3');   %
Tiempo, Aceleración
grid on
```

```
% EJEMPLO 1
% t = [0, 3, 8, 12, 16, 22, 33];
% q = [0, 4, 12, 12, 8, 20, 24];
% npuntos = 300;
% trayectoria = interpolador_cubico_dibujo(t,npuntos,q)
% EJEMPLO 2
% t = [0, 3, 8, 12, 16, 22, 33];
```

```
% q = [0, 4, 12, 12, 8, 20, 24];
% qp = [0,-1, 3, 0, 3,-3, 0];
% npuntos = 300;
% trayectoria = interpolador_cubico_dibujo(t,npuntos,q,qp)

function trayectoria = interpolador_cubico_dibujo(t,npuntos,q,qp)
n=length(q);    % numero de intervalos
clf
hold on
i=1;
if nargin==3
    P=interpolador_cubico(t,q,qp);
else
    P=interpolador_cubico(t,q);
end

for intervalo=1:n-1
    ti =    P.ti(intervalo);
    tf =    P.tim1(intervalo);
    a  =    P.a(intervalo);
    b  =    P.b(intervalo);
    c  =    P.c(intervalo);
    d  =    P.d(intervalo);
    inc=(tf-ti)/npuntos;
    for tt=ti:inc:tf
        tti(i)=tt;
        qt(i)=a+b*(tti(i)-ti)+c*(tti(i)-ti)^2+d*(tti(i)-ti)^3;
        qpt(i)=b+2*c*(tti(i)-ti)+3*d*(tti(i)-ti)^2;
        qppt(i)=2*c+6*d*(tti(i)-ti);
        plot(tti(i),qt(i),'ok');
        i=i+1;
    end

end

plot(t,q,'-og')
grid
hold off

% ----------------------------- RESULTADOS --------------------------------
trayectoria.t = tti
trayectoria.q = qt
trayectoria.qp = qpt
trayectoria.qpp = qppt
end
```

```
% EJEMPLO 1
% t = [0, 3, 8, 12, 16, 22, 33];
% q = [0, 4, 12, 12, 8, 20, 24];
% qp = [0, 0, 0, 0, 0, 0, 0];
% trayectoria = interpolador_cubico(t,q,qp)
% EJEMPLO 2
% t = [0, 3, 8, 12, 16, 22, 33];
% q = [0, 4, 12, 12, 8, 20, 24];
% qp = [0,-1, 3, 0, 3,-3, 0];
% trayectoria = interpolador_cubico(t,q,qp)
function trayectoria = interpolador_cubico(t,q,qp)
n=length(q);
if n~=length(t)
    error('ERROR en i_cubico: Las dimensiones de q y t deben ser iguales')
end
if nargin~= 3    % qd no definida. La obtiene  segun [6.8]
   qp(1)=0;
   qp(n)=0;
  for i=2:n-1
      if (sign(q(i)-q(i-1))==sign(q(i+1)-q(i)))...
              |q(i)==q(i+1)...
              |q(i-1)==q(i)
          qp(i)=0.5*((q(i+1)-q(i))/(t(i+1)-t(i))+(q(i)-q(i-1))/(t(i)-t(i-
1)));
      else
          qp(i)=0;
      end
  end
end
% obtiene los coeficientes de los polinomios
    for i=1:n-1
        ti=t(i);
        tii=t(i+1);
        if tii<=ti
            error ('ERROR en i_cubico. Los tiempos deben estar ordenados:
t(i) debe ser < t(i+1)')
        end
        T=tii-ti;
        TT(:,i)=[ti;tii];
        a(i)=q(i);
        b(i)=qp(i);
        c(i)= 3/T^2*(q(i+1)-q(i)) - 1/T  *(qp(i+1)+2*qp(i));
        d(i)=-2/T^3*(q(i+1)-q(i)) + 1/T^2*(qp(i+1)  +qp(i));
    end

    % ---------------------- RESULTADOS ---------------------------------
    trayectoria.ti = TT(1,:);
    trayectoria.tim1 = TT(2,:);
    trayectoria.a = a;
    trayectoria.b = b;
    trayectoria.c = c;
    trayectoria.d = d;
```

```
% EJEMPLO 1
% t = [0, 3, 8, 12, 16, 22, 33];
% q = [0, 4, 12, 12, 8, 20, 24];
% npuntos = 300;
% trayectoria = interpolador_cubico_dibujo(t,npuntos,q)
% EJEMPLO 2
% t = [0, 3, 8, 12, 16, 22, 33];
% q = [0, 4, 12, 12, 8, 20, 24];
% qp = [0,-1, 3, 0, 3,-3, 0];
% npuntos = 300;
% trayectoria = interpolador_cubico_dibujo(t,npuntos,q,qp)

function trayectoria = interpolador_cubico_dibujo(t,npuntos,q,qp)
n=length(q);    % numero de intervalos
clf
hold on
i=1;
% Obtiene los coeficientes de los splines cubicos
% [ti,tf,a,b,c,d] para cada intervalo
if nargin==3
    P=interpolador_cubico(t,q,qp);
%       P=i_cubico(q,t,qp);
else
    P=interpolador_cubico(t,q);
%       P=i_cubico(q,t);
end
for intervalo=1:n-1
    ti = P.ti(intervalo);
    tf = P.tim1(intervalo);
    a  = P.a(intervalo);
    b  = P.b(intervalo);
    c  = P.c(intervalo);
    d  = P.d(intervalo);

    inc=(tf-ti)/npuntos;
    for tt=ti:inc:tf
%         tti(i)=tt;
%         qt=a+b*(tt-ti)+c*(tt-ti)^2+d*(tt-ti)^3;
%         qpt=b+2*c*(tt-ti)+3*d*(tt-ti)^2;
%         qppt=2*c+6*d*(tt-ti);
%         plot(tt,qt,'k');
        tti(i)=tt;
        qt(i)=a+b*(tti(i)-ti)+c*(tti(i)-ti)^2+d*(tti(i)-ti)^3;
        qpt(i)=b+2*c*(tti(i)-ti)+3*d*(tti(i)-ti)^2;
        qppt(i)=2*c+6*d*(tti(i)-ti);
        plot(tti(i),qt(i),'ok');
        i=i+1;
    end
```

```
% Trayectoria_interpolador_cubica.m
%  clear, clc
% Ejemplo
px = [-100 100 100 -100 -100 ];
py = [50 50 250 250 50];
pz = [100 100 100 100 100];
npuntos=3; n=15; u=0;
r=length(px);
% Creo diviciones en los puntos para tener mayor resolución
for k=1:r-1
sum_x(k) = abs((px(k+1)-px(k))/n);
sum_y(k) = abs((py(k+1)-py(k))/n);
sum_z(k) = abs((pz(k+1)-pz(k))/n);
for j=1:n
    u=u+1;
    if px(k+1)> px(k)
        pxi(u) = px(k)+sum_x(k)*j;
    else
        pxi(u) = px(k)-sum_x(k)*j;
    end
    if py(k+1)> py(k)
        pyi(u) = py(k)+sum_y(k)*j;
    else
        pyi(u) = py(k)-sum_y(k)*j;
    end
    if pz(k+1)> pz(k)
        pzi(u) = pz(k)+sum_z(k)*j;
    else
        pzi(u) = pz(k)-sum_z(k)*j;
    end
    ti(u)=u-1;
end
end
w=length(pxi)
CODO = -1;  % Codo arriba
for i=1:w
    P=[pxi(i);pyi(i);pzi(i)];
    Q(i,:)=Cinematica_inversa_3_gdl_paralelo(CODO,P);
end
q1=Q(:,1)';
q2=Q(:,2)';
q3=Q(:,3)';
% Calculo trayectorias
trayectoria1 = interpolador_cubico_dibujo(ti,npuntos,q1,1);
trayectoria2 = interpolador_cubico_dibujo(ti,npuntos,q2,1);
trayectoria3 = interpolador_cubico_dibujo(ti,npuntos,q3,1);
q1=trayectoria1.q;
q2=trayectoria2.q;
q3=trayectoria3.q;
M(:,1)=q1;
M(:,2)=q2;
M(:,3)=q3;
tp=length(q1);
f1 = figure(1); set(f1,'Color',[1, 1, 1]);
```

```
writerObj = VideoWriter('Video_animar_iniciales');            % LINEAS PARA
VIDEO
writerObj.FrameRate = 30;                                     % LINEAS PARA VIDEO
open(writerObj);                                              % LINEAS PARA VIDEO
nv = 3; % numero de veces que añade cada frame al video % LINEAS PARA VIDEO
for i = 1:tp
    q_cubica(i,1:3) = M(i,:);
    qbrazo =  M(i,:);
    PARALELO = cin_directa_3_gdl_paralelo(qbrazo);
    xyz(i,1:3)=PARALELO.A03(1:3,4);
    % Animación del brazo
    figure(1); dibujar_robot(PARALELO); hold off
    axis([-250 250 -250 250 0 400]); grid on;
end
% ------------ RESULTADOS GRAFICOS --------------------
figure(1); dibujar_robot(PARALELO);
plot3(xyz(:,1),xyz(:,2),xyz(:,3),'o'); hold on;
axis([-250 250 -250 250 0 400]); grid on;
% ---------------- LINEAS VIDEO -------------------------
   for nv = 1:nv                                            % LINEAS PARA VIDEO
        frame = getframe(f1);                               % LINEAS PARA VIDEO
        writeVideo(writerObj,frame);                        % LINEAS PARA VIDEO
   end
figure(2); plot3(xyz(:,1),xyz(:,2),xyz(:,3),'o'); hold on;
% Esta linea evita el error de incremento del ultimo eslabón en simulink
q_cubica(tp+1,3)=q_cubica(tp,3);
q_cubica(tp+1,2)=q_cubica(tp,2);
q_cubica(tp+1,1)=q_cubica(tp,1);
tiempo_cubica=linspace(0,tp/8,tp);
tiempo_cubica(tp+1)=tiempo_cubica(tp)+tiempo_cubica(2);
% ---------------------------- VARIABLES ARTICULARES -----------------------
-------
q1_spline_cubico = timeseries(q_cubica(:,1), tiempo_cubica, 'name',
'Position');
q2_spline_cubico = timeseries(q_cubica(:,2), tiempo_cubica, 'name',
'Position');
q3_spline_cubico = timeseries(q_cubica(:,3), tiempo_cubica, 'name',
'Position');
```

```
% Variables_controlador.m
clear, clc;
% Parametros Motor
DCMotor_R = 4.9993815260278314;
DCMotor_L = 3.164547121747764e-04;
DCMotor_K = 0.477771340295273;
DCMotor_Kf = 0.921434142151143;
% Parametros PID
Hombro_Kd = -3.5815;
Hombro_Ki = 2.2752;
Hombro_Kp = 270.3634;
Hombro_N = 100;
Brazo_Kd = -3.5815;
Brazo_Ki = 2.2752;
Brazo_Kp = 270.3634;
Brazo_N = 100;
Antebrazo_Kd = -3.5815;
Antebrazo_Ki = 2.2752;
Antebrazo_Kp = 270.3634;
Antebrazo_N = 100;
```