UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño de un banco de pruebas para un robot serial adaptable

TESIS

INGENIERO MECATRÓNICO

P R E S E N T A N

JESÚS CASTAÑEDA ESPINOSA
ERICK EDUARDO PEDRO MENDOZA

DIRECTOR DE TESIS

DR. VÍCTOR JAVIER GONZÁLEZ VILLELA

Ciudad Universitaria, Cd. Mx., 2016

Agradecimientos

Agradecemos a la Universidad Nacional Autónoma de México por habernos brindado una excelente formación profesional.

A los profesores de la Facultad de Ingeniería por habernos brindado los conocimientos necesarios para lograr conseguir una formación profesional.

Al Dr. Víctor Javier Gonzáles Villela, por dirigir esta tesis de manera exitosa y permitirnos comprender la importancia de la investigación en Ingeniería.

A los miembros del jurado, M.I. Ana Marissa Juárez Mendoza, M.A. Luis Yair Bautista Blanco, M.I. Erik Peña Medina, Ing. Noé Alfredo Martínez Sánchez, quienes nos brindaron su apoyo durante esta etapa como estudiantes de ingeniería e impulsarnos a mejorar nuestra formación personal y profesional.

Agradecemos en lo que corresponde a la DGAPA, por el apoyo brindado para la realización de este trabajo, a través del proyecto UNAM-DGAPA-PAPIIT IN117614: “Robótica intuitiva, adaptable, reactiva, híbrida y móvil aplicada al servicio, el rescate y la medicina”

Índice

Resumen

En este trabajo se muestra el diseño y la construcción de un prototipo funcional de un robot serial adaptable, se ve paso a paso la selección de actuadores, sensores, microcontroladores, materiales y herramientas que se emplean y se consideran durante el proceso de diseño y construcción de dicho robot. También se muestra el análisis de la cinemática directa e inversa, así como el análisis de cargas estáticas que soportan los actuadores.

Es posible apreciar el diseño electrónico, donde se menciona que la comunicación entre los servomotores Dynamixel y SIMULINK se da mediante protocolo de comunicación RS-485 y entre la tarjeta MD25 y SIMULINK se da mediante comunicación I2C, así mismo se detallan las conexiones pertinentes y necesarias que requieren todos y cada uno de los componentes empleados en el robot serial adaptable para que éste funcione correctamente.

También se puede consultar la metodología de diseño de software empleada para el diseño de dos módulos en SIMULINK. El primer módulo (SIMULINK-Dynamixel) permite enviar los valores de posición deseada y leer los valores de la posición actual, velocidad, porcentaje de carga, sentido de giro, voltaje y temperatura de cada uno de estos, además, dicho módulo es capaz de controlar hasta 255 servomotores. El segundo módulo (SIMULINK-MD25) ayuda a enviar y recibir datos de la tarjeta controladora de motores MD25, para enviar los valores de velocidad y leer los valores de los encoder, voltaje de operación y corriente consumida de cada uno de los motores de CD conectados dicha tarjeta, además, este módulo es capaz de controlar hasta 8 tarjetas MD25.

Se muestra como se emplean las herramientas SIMULINK-MD25 y SIMULINK-Dynamixel para lograr utilizar al robot serial adaptable como un banco de pruebas que permite probar teorías y realizar experimentos que ayudan a entender el fenómeno de adaptabilidad, del cual se está investigando en el *Mechatronic Research Group* (MRG) de la Facultad de Ingeniería de la UNAM.

Finalmente se aprecian los resultados de las pruebas realizadas con los módulos en el robot serial adaptable, se muestran las gráficas que detallan el comportamiento real del robot mientras este sigue una trayectoria de una elipse: demostrando que los módulos funcionan correctamente y pueden ser empleados para controlar cualquier motor DC empleando la tarjeta MD25 y cualquier servomotor que utilice comunicación RS-485, y además se comprueba que el robot serial adaptable tiene diversos comportamientos.

Capítulo 1

Generalidades

1.1 Introducción

Uno de los problemas en que se encuentra la robótica actual es el diseñar robots que puedan desempeñar muchas funciones. La mayoría de los robots actuales sólo pueden realizar una tarea o múltiples tareas similares como es el caso de los robots industriales.(Prieto, 2003)

Los robots industriales han tenido un crecimiento acelerado, pues se han instalado en muchas de las industrias manufactureras, en particular en la automotriz. En el año 2000 se estimaba que había unas 800,000 unidades operativas en el mundo, este número de robots cubría, en gran medida, todas aquellas instalaciones en las que el uso del robot es económicamente rentable, teniendo una tasa de crecimiento escasa en los últimos años. (Barrientos, 2002)

A mediados de los años 80 surgieron otras aplicaciones para los robots, sin relación con la manufactura, donde se buscaba encontrar ventajas en el uso de robots, distintas al aumento de la productividad. Estas nuevas aplicaciones se caracterizan por desarrollarse fuera del ambiente estructurado de una fábrica (donde casi todo está previsto), es decir, actividades en donde no se restrinja la interacción del robot con piezas o herramientas dentro de un entorno construido o adaptado para que los robots trabajen sin ningún percance. Barrientos (2002)

Las principales limitaciones de los robots industriales son: su movilidad, su capacidad de interactuar con el ser humano y su capacidad de adaptarse a entornos no estructurados con condiciones ambientales cambiantes, es por ello que surge el desarrollo de los robots de servicio. (Barrientos, 2002) El término de robot de servicio apareció a finales de los años 80 como una necesidad de satisfacer las limitaciones anteriormente descritas.

La Federación Internacional de Robótica, organismo que coordina las actividades en esta área tecnológica de los países con mayor nivel de desarrollo, define un robot de servicio como: “Un robot que opera de manera automática o semiautomática para realizar servicios útiles al bienestar de los humanos o a su equipamiento, excluyendo las operaciones de fabricación” (Aracil, Balaguer, & Armada, 2008)

En 1995 fue creado el Comité Técnico de Robots de Servicio (*Technical Committee on Service Robots)* por la Sociedad de Robótica y Automatización de la IEEE (*IEEE Robotics and Automation Society)*. Aracil et al. (2008)

Según (Aracil et al., 2008), en el año 2000 el *Technical Committee on Service Robots* definió las principales áreas de aplicación de los robots de servicio, éstas se pueden dividir en dos grandes grupos:

- Sectores productivos no manufactureros: edificación, agricultura, naval, minería, medicina, etc.
- Sectores de servicio: asistencia personal, limpieza, vigilancia, educación, entretenimiento, etc.

(Barrientos, 2002) Estimó que en el año 2002 ya habían 1200 robots de servicio operativos, distribuidos en las actividades antes mencionadas, como se muestra en la Imagen 1.1.

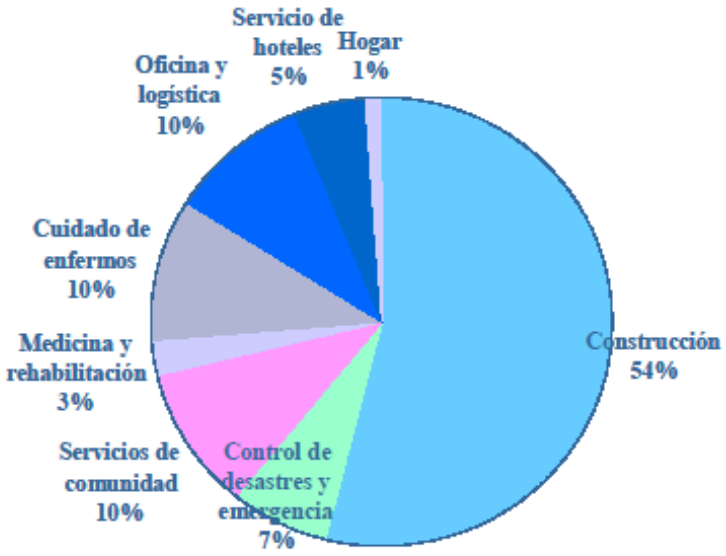


Imagen 1.1 Distribución por aplicaciones de los robots de servicio.

Otra solución a la problemática anteriormente descrita es: el diseño de robots modulares, que surgen de la imitación de los sistemas biológicos. Se divide al robot en múltiples partes consideradas como un módulo, haciendo una analogía con las células, de forma que un simple módulo por sí mismo no puede desempeñar una tarea, pero en conjunto puede desempeñar diversas tareas.(Prieto, 2003)

Existen dos grandes clasificaciones de los robots modulares, las cuales propone (Prieto, 2003) como:

- Lattice (red): Su principal característica es que las articulaciones en éste tipo de robots son prismáticas.
- Chain (cadena): Sus articulaciones son por lo general rotacionales, produciendo ángulos. Actúan de manera similar al movimiento de una oruga o serpiente.

Algunas de las características principales de los robos modulares, según (Prieto, 2003) son:

- Los módulos deben ser pequeños y simples, en función del tamaño físico por módulo y el número de componentes.
- Utilizan diodos infrarrojos para intercambiar información entre los módulos.
- Los módulos se unen principalmente mediante electroimanes IB magnet pero también pueden ser unidos mecánicamente.

Otra forma de solucionar dicho problema es la propuesta presentada en este trabajo, definida como ***robótica serial adaptable***, la cual consiste en variar la longitud de los eslabones del robot, con la finalidad de que éste tenga una infinidad de configuraciones posibles, dándole la cualidad de poder adaptarse a diversos ambientes. Esta idea surge de la observación, de cómo es que los seres vivos somos capaces de adaptarnos a un ambiente.

La teoría de Darwin nos dice que la adaptación ocurrirá con el tiempo, teniendo como resultado la evolución de nuevas especies, la adaptación y cambio en las existentes, con la finalidad de sobrevivir a diversos cambios ambientales, esto es, para obtener la mayor oportunidad de que los genes perduren en las generaciones futuras, a esta adaptación de las especies a través del tiempo de le llama *adaptación filogenética*. Los cambios que ocurren dentro del tiempo de vida de un individuo de cualquier especie para adaptarse a su ambiente, se le conoce como *adaptación ontogénica* (Birch, Ann, & Sheila, 1998).

Actualmente, se cuenta con información referente a los robot modulares o reconfigurables que menciona (Prieto, 2003), como una solución a las limitaciones de los robot industriales, anteriormente mencionadas. Con base en lo anterior es posible tener un punto de partida para poder realizar el diseño y construcción de un robot serial adaptable.

Con la finalidad de obtener información sobre cómo la variación de dimensiones puede o no afectar a la cinemática y dinámica de un robot serial adaptable, así como también ver cómo afecta la demanda de potencia en sus actuadores mientras sus dimensiones cambian, por ello se diseñó un banco de pruebas que nos permita reproducir los fenómenos, anteriormente descritos, en torno al tema, con la finalidad de lograr establecer una base de conocimiento para el desarrollo posterior de este tipo de robots, capaces de adaptarse a las diferentes tareas que se le den y que además optimice los recursos de consumo eléctrico y desgaste mecánico, al demandar menos potencia a sus actuadores.

Lo anterior permitirá aplicar dicho conocimiento en distintos campos de la ingeniería, así como en muchas otras disciplinas con fines académicas y áreas de desarrollo tecnológico, generando en un futuro la posibilidad de crear un campo de estudio entorno a la robótica serial adaptable.

1.2 Antecedentes

La robótica es una ciencia o rama de la tecnología, que estudia el diseño y construcción de máquinas capaces de desempeñar tareas realizadas por el ser humano. ((UNSAAC), 2006) También se puede decir que la robótica es la ciencia que estudia el diseño y la implementación de robots, conjugando múltiples disciplinas, como la mecánica, la electrónica, la informática, la inteligencia artificial y la ingeniería de control, entre otras. (Costas, 2012)

Existen tres leyes de la robótica descritas en (Saha, 2010) las cuales son básicas y fundamentales para la coexistencia entre seres humanos y robots, las cuales son:

1. Un robot no debe dañar a un ser humano ni, por su inacción, dejar que un ser humano sufra daño.
2. Un robot debe obedecer las órdenes que le son dadas por un ser humano, excepto si estas entran en conflicto con la primera ley.
3. Un robot debe proteger su propia existencia, a menos que ésta entre en conflicto con las dos primeras leyes.

Más tarde, (Fuller, 1991) introdujo una cuarta ley que dice:

4. Un robot podrá tomar el trabajo de un ser humano, pero no debe dejar a ésta persona sin empleo.

El término *robot* aparece por primera vez en 1921, en la obra teatral R.U.R. (Rossum´s Universal Robots) del novelista y autor dramático checo Karel Capek en cuyo idioma la palabra "*robota*" significa fuerza del trabajo o servidumbre.(Baturone, 2005)

El término *robot* lo define de manera formal la Organización Internacional para la Estandarización (ISO) (Saha, 2010), como un manipulador *multifuncional reprogramable*, capaz de mover materiales, piezas, herramientas o dispositivos especiales, a través de movimientos variables programados, para el desempeño de tareas diversas.

Entre los robots considerados de más utilidad en la actualidad se encuentran los *robots industriales*. La Federación Internacional de Robótica (IFR, *International Federation of Robotics*) define a un Robot Industrial como:

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

La anterior definición se debe entender que la reprogramación y la multifuncionalidad se consiguen sin modificaciones físicas del robot, lo cual se limita a que pueden realizar diversas tareas siempre y cuando queden dentro del espacio de trabajo al que fueron configurados.

De acuerdo con la definición anterior, adoptada por la Federación Internacional de Robótica, en conjunto con la N***orma ISO/TR 8373***, se puede definir a un robot manipulador industrial de la siguiente manera:

Un *robot manipulador industrial* es una máquina manipuladora con varios grados de libertad controlada automáticamente, reprogramable y de múltiples usos, pudiendo estar en un lugar fijo o móvil para su empleo en aplicaciones industriales.(Kelly & Santibáñez, 2003)

Un manipulador es un brazo mecánico articulado formado de eslabones conectados a través de uniones o articulaciones que permiten un movimiento relativo entre dos eslabones consecutivos. El movimiento de cada articulación puede ser trasnacional, rotacional o una combinación de ambos. (Kelly & Santibáñez, 2003)

En la Imagen 1.2 se muestran los principales elementos básicos en un robot, como son: sus eslabones y las articulaciones, anteriormente mencionadas.

Imagen 1.2 Elementos básicos de un manipulador.

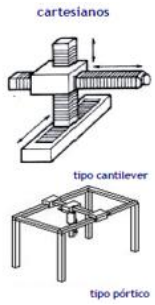
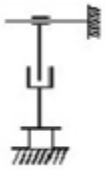
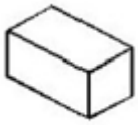
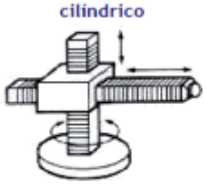
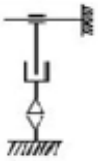
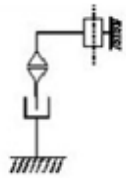
(Baturone, 2005) Considera cinco tipos de clasificación de los robots según su geometría, dependiendo su estructura mecánica, pueden ser:

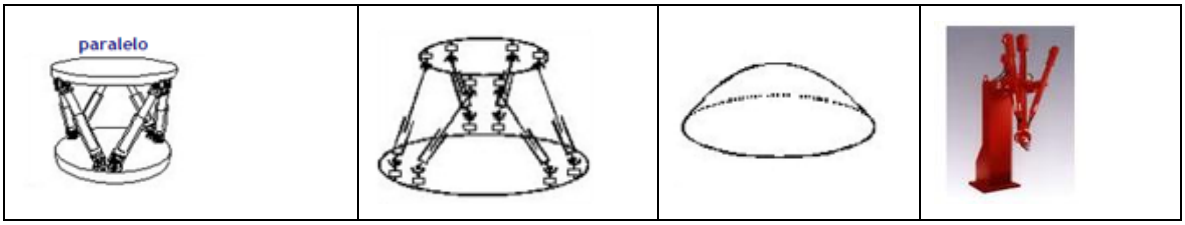
- **Cartesiano**, cuyo posicionamiento en el espacio se lleva a cabo mediante articulaciones lineales.
- **Cilíndrico**, con una articulación rotacional sobre una base y articulaciones lineales para el movimiento en altura y en radio.
- **Polar o Esférico**, con tres articulaciones rotacionales que cuenta con dos articulaciones rotacionales y una lineal.
- **Angular**, tiene tres articulaciones de rotación.
- **Mixto**, que posee varios tipos de articulaciones, combinaciones de las anteriores. Es destacable la configuración **SCARA** (*Selective Compliance Assembly Robot Arm*).

- **Paralelos,** tienen la característica de que poseen brazos con articulaciones prismáticas o rotacionales concurrentes. (González, 2004) considera que los robots paralelos también entran dentro de esta clasificación de robots seriales.

En la Tabla 1.1 se muestran los tipos de robots según su geometría.

Tabla 1.1 Configuraciones típicas de los robots industriales.

Configuración geométrica	Estructura cinemática	Espacio de trabajo	Ejemplo
			
			
			
			
			



Es importante definir algunos conceptos antes de mencionar cuales son los principales parámetros que caracterizan a los robots industriales, como cadena cinemática, grados de libertad, etc.

- Cadena cinemática:

(González, 2004) Al conjunto de eslabones y articulaciones se denomina *cadena cinemática*. Se dice que una cadena cinemática es abierta si cada eslabón se conecta mediante articulaciones exclusivamente al anterior y al siguiente, exceptuando el primero, que se suele fijar a un soporte, y el último, cuyo extremo final queda libre. A éste se puede conectar un *elemento terminal* o *actuador final:* una herramienta especial que permite al robot de uso general realizar una aplicación particular, que debe diseñarse específicamente para dicha aplicación: una herramienta de sujeción, de soldadura, de pintura, etc. El punto más significativo del elemento terminal se denomina *punto terminal (PT)*. En el caso de una pinza, el punto terminal vendría a ser el centro de sujeción de la misma.

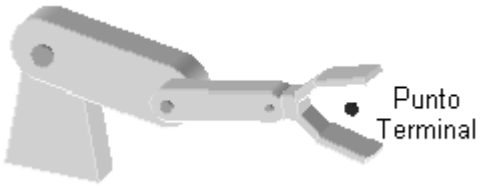


Imagen 1.3 Punto terminal de un manipulador.

- Grados de libertad (GDL):

Se denomina *grado de libertad* a cada una de los movimientos independientes que puede realizar cada articulación con respecto a la anterior. El número de grados de libertad del robot viene dado por la suma de los grados de libertad de las articulaciones que lo componen. (Barrientos, Peñín, Balaguer, & Aracil, 2007). En la Imagen 1.4 se muestran el número de grados de libertad que están presentes en los diferentes tipos de junta que existen.

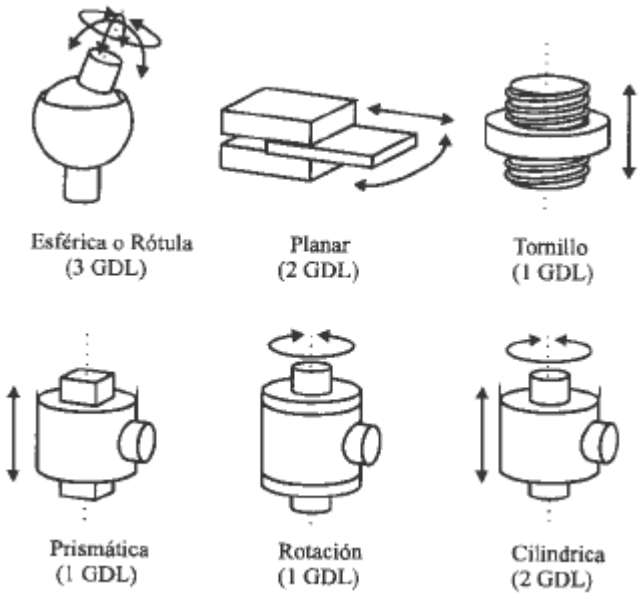


Ilustración 1.4 Tipos de juntas y su número de grados de libertad.

Normalmente, en cadenas cinemáticas abiertas, cada par eslabón-articulación tiene un solo grado de libertad, ya sea de rotación o de traslación. Pero una articulación podría tener dos o más GDL que operan sobre ejes que se cortan entre sí.

Imagen 1.5 Grados de libertad de un manipulador.

Los principales parámetros que caracterizan a los robots industriales son:

- *Número de grados de libertad:* Es el número total de grados de libertad de un robot, dado por la suma de GDL de las articulaciones que lo componen.
- *Espacio de accesibilidad o espacio de trabajo*: Es el conjunto de puntos del espacio accesibles al punto terminal (PT), que depende de la configuración geométrica del manipulador. Un punto del espacio se dice totalmente accesible si el PT puede situarse en él en todas las orientaciones que permita la constitución del manipulador

y se dice parcialmente accesible si es accesible por el PT pero no en todas las orientaciones posibles.

- *Capacidad de posicionamiento del punto terminal*: Se concreta en tres magnitudes fundamentales: resolución espacial, precisión y repetitividad, que miden el grado de exactitud en la realización de los movimientos de un manipulador al realizar una tarea programada.
- *Capacidad de carga:* Es el peso que puede transportar el elemento terminal del manipulador. Es una de las características que más se tienen en cuenta en la selección de un robot dependiendo de la tarea a la que se destine.
- *Velocidad:* Es la máxima velocidad que alcanzan el Punto Terminal y las articulaciones.

Con los términos que se emplearán a lo largo de éste escrito ya definidos, ahora es posible entrar a la explicación del desarrollo de la presente tesis.

1.3 Planteamiento del problema

En la robótica actual, los robots tienen ciertas restricciones mecánicas, eléctricas, de programación, entre otras, que de alguna u otra forma afectan su funcionamiento y provocan que éstos sean limitados, es decir, que prácticamente estén diseñados para realizar una tarea en específico y no múltiples tareas, dando como resultado que sea necesario tener un robot para cada tarea.

Como parte de una solución a la problemática anteriormente descrita, se planteó la posibilidad de trabajar en la realización de un proyecto enfocado al área de la robótica, con la finalidad de crear una manera alternativa y conveniente de diseño de un robot adaptable, donde éste debe ser capaz de cambiar sus dimensiones. Uno de los requerimientos, es el poder leer valores en tiempo real, del robot adaptable, mientras éste ejecuta una tarea en específico, con la finalidad de generar una base de conocimiento experimental que nos permita corroborar ciertas hipótesis con respecto a esta área de la robótica adaptable, que sirva para trabajos de investigación futuros.

En este proyecto hay algunos inconvenientes inherentes a la elaboración del diseño de un robot adaptable, como:

a) Actualmente no se cuenta con información que detalle cómo se hacen, cómo funcionan y mucho menos se sabe con certeza cuál es su comportamiento cuando se encuentra en movimiento un robot adaptable.

b) La tecnología con la que se cuenta (motores, servomotores) es limitada e interfiere con el diseño del robot adaptable, pues provoca que el diseño del robot se tenga que amoldar a los motores y servomotores y no al revés. Sin embargo, en los capítulos posteriores se muestra cómo fue posible la elaboración del diseño del mismo.

1.4 Justificación

Se esperan grandes beneficios de este proyecto, porque forma parte de un tema de actualidad y presenta cualidades de algunos trabajos de investigación que están realizando en otras universidades, tal es el caso de los robots modulares del *Massachusetts Institute of Technology* (MIT). Tanto el diseño de los robots modulares como el diseño del robot adaptable, presentan una nueva alternativa de solución a la problemática de los robots actuales, que es el no poder desempeñar múltiples tareas, con la diferencia de que los robots modulares que fueron diseñados son: pequeños eslabones que se unen entre sí mediante una articulación magnética motorizada, que les permite a los eslabones cambiar de posición, forma y tamaño.

Ésta nueva propuesta de diseño podrá ser utilizada en diversas áreas industriales, tales como: empresas manufactureras que utilicen manipuladores industriales para llevar acabo sus procesos, pues el robot adaptable al poder variar sus dimensiones, tendrá la capacidad de poder ocupar un menor espacio de trabajo, operando a una misma velocidad de un robot industrial o incluso mejor. También podrá ser utilizado con fines educativos y de investigación, con el propósito de seguir desarrollando nuevas mejoras.

Tras el desarrollo de este proyecto, también se podrán resolver las problemáticas de espacio, pues en la actualidad lo que se busca es la optimización de los mismos y con este proyecto podremos lograr que un robot pueda cambiar sus dimensiones a tal grado de que pueda ser tan grande o tan pequeño como sea posible.

Los resultados a obtener del proyecto son: el de diseño y la fabricación de un robot adaptable, así como la elaboración de una herramienta diseñada en SIMULINK para enviar y recibir datos de servomotores Dynamixel, mediante comunicación RS-485, y datos de motores DC controlados con tarjetas MD25, así también se espera realizar una prueba que permita hacer una comparativa del robot cuando éste ejecuta una tarea en dos diferentes configuraciones del mismo.

La trascendencia del actual proyecto, marcará un gran desarrollo en este tipo de robots adaptables, en un futuro estos robots pueden llegar a tener estructuras mucho más complejas, con la capacidad de no sólo cambiar de dimensiones sino que también dispongan de la inteligencia, mecánica y programable, para poder cambiar incluso de forma y apariencia.

1.5 Objetivos del proyecto

1. Diseñar un robot adaptable capaz de hacer variar las dimensiones longitudinales de sus eslabones, de al menos 3 grados de libertad, empleando juntas rotacionales y prismáticas para su elaboración.
2. Diseñar una herramienta en SIMULINK que ayude a enviar y recibir datos a Servomotores Dynamixel con comunicación RS-485, donde se envíen los valores de posición deseada y se lean la posición actual, velocidad, porcentaje de carga, sentido de giro, voltaje y temperatura de cada uno de los servomotores conectados, además, la herramienta debe ser capaz de controlar hasta 255 servomotores.
3. Diseñar una herramienta en SIMULINK que ayude a enviar y recibir datos a la tarjeta controladora de motores MD25, donde se envíen los valores de velocidad y se lean los valores de los encoder, voltaje de operación y corriente consumida de cada uno de los motores de CD conectados a la tarjeta, además, la herramienta debe ser capaz de controlar hasta 8 tarjetas MD25.
4. Emplear las herramientas SIMULINK-MD25 y SIMULINK-Dynamixel para utilizar al robot serial adaptable como un banco de pruebas que permita probar teorías y realizar experimentos que ayuden a entender el fenómeno de adaptabilidad, del cual se está investigando en el *Mechatronic Research Group* (MRG) de la Facultad de Ingeniería de la UNAM.

1.6 Hipótesis

Existe un banco de pruebas que nos permite controlar las dimensiones predefinidas de los eslabones de un robot serial adaptable, leer las variables de los sensores y estimar la potencia consumida en sus motores, cuando este ejecuta una trayectoria.

1.7 Propósito

El propósito del banco de pruebas es, obtener información necesaria de los actuadores de un robot adaptable, para poder estimar la potencia consumida o necesaria para que éste realice una tarea dada, siendo capaz de seguir una trayectoria con una determinada o diversas configuraciones.

La finalidad es estimar las longitudes de los eslabones en las cuales al robot adaptable le es más conveniente (Fácilmente) realizar una tarea o seguir una trayectoria dada para que los actuadores trabajen con un consumo menor de potencia. Se propone variar sus dimensiones para determinar la configuración o configuraciones más adecuadas (donde a los actuadores no les cuesta trabajo moverse) para el robot adaptable. Como parte del experimento se obtendrán muestras individuales de cada actuador para así estimar particularmente que junta es la que consume más potencia y con base en ello poder encontrar una configuración adecuada para minimizar el trabajo que realice.

La ubicación o el ambiente donde se desenvolverá el banco de pruebas están dentro del Centro de Ingeniería Avanzada (CIA) de la Facultad de Ingeniería y su objetivo será probar las teorías propuestas sobre el robot serial adaptable.

Las limitaciones presentes en el proceso es que inicialmente se tiene un robot que sólo trabaja en un espacio de trabajo determinado (Limitado), ya que este no cuenta con una base móvil. Se restringe su movilidad respecto de los grados de libertad que el robot tiene, así como también por su capacidad de par que pueden proporcionar los actuadores empleados, otra restricción dentro del análisis a considerar es la influencia de la gravedad aplicada al robot, pues no sólo le afecta cuando está en estado estático, sino que se ve más afectado cuando éste realiza una trayectoria, específicamente, cuando los eslabones llevan una trayectoria ascendente porque tienden a bajar más rápido y a subir más lento.

Capítulo 2

Diseño del robot adaptable.

2.1 Introducción

El diseño en ingeniería se genera a partir del proceso de aplicar diversas técnicas y principios científicos con el objetivo de dar una solución a un determinado problema, donde generalmente ésta suele ser la más óptima.

En este capítulo se muestra la metodología de diseño que se utilizó para definir las formas y tamaños presentes en el robot adaptable, se proponen los materiales más óptimos a emplear para su elaboración. Se presentan los elementos principales que conforman a los eslabones del robot adaptable y se detallan las bases sobre las cuales se planteó el diseño, en cuanto a disponibilidad de materiales y recursos de tecnología. También, se hace la selección de los actuadores, controladores, protocolo de comunicación y demás herramientas indispensables para el correcto funcionamiento del robot adaptable.

2.2 Arquitectura del robot adaptable

El mecanismo propuesto, es un manipulador serial que en cada uno de sus eslabones cuenta con juntas rotacionales y prismáticas, siendo estas últimas las que le dan, precisamente, la cualidad de ser un robot adaptable, pues permite que los eslabones del robot cambien de dimensiones al ir variando la longitud de los mismos. En la Imagen 2.1 se muestran los elementos básicos, antes mencionados, presentes en el robot adaptable.

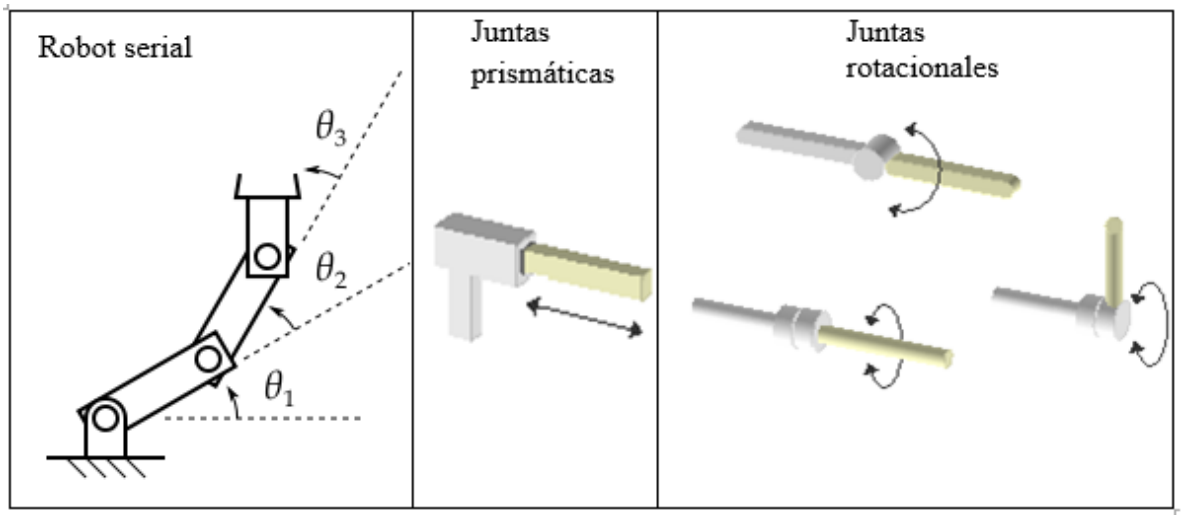


Imagen 2.1 Elementos que componen al robot adaptable.

2.3 Requerimientos y especificaciones

Se planteó la necesidad de diseñar una propuesta de un prototipo funcional de un robot serial adaptable, cuya configuración ofrezca poder operar en un mayor espacio de trabajo ocupando menor espacio en su estado de reposo (estado inicial), mediante la utilización de juntas prismáticas que permitan la variación de las dimensiones de sus eslabones. Además, dicho robot adaptable debe ser capaz de poder encontrar una configuración adecuada, para dar una o varias soluciones a una tarea dada, utilizando la configuración más óptima para que éste pueda trabajar utilizando la menor potencia posible de sus actuadores debido a la variación de la dimensión de sus eslabones.

2.4 Diseño conceptual

El robot adaptable está formado principalmente por 2 eslabones, juntas rotacionales, juntas prismáticas y la base sobre la cual está colocado. Se propuso que el diseño de las piezas que conforman al robot adaptable fueran de geometrías simples, con la finalidad de facilitar el ensamble de las mismas, que en su mayoría son de acrílico cortado con láser (Acrilico, 2016).

Para tener una buena unión entre las piezas y así evitar que éstas se despegaran o se movieran, se emplearon dos metodologías utilizadas cuando se realizan trabajos de carpintería, donde se requiere tener una buena unión entre dos trozos de madera, dichas metodologías se denominan; ensamble de caja y espiga (Albano, 2016) y ensamble de cola de milano (Albano, 2012).

El método de caja y espiga se emplea en general cuando se tienen que unir piezas con un ángulo de 90°. Existen muchas maneras de lograr el ensamble y la más ideal es hacer que el extremo de una de las piezas encaje en un agujero realizado en otra. El extremo de la primera pieza se le llama espiga y esta normalmente se corta de un tamaño más pequeño que el del agujero (caja) en el que se va a introducir. Este tipo de acoplamiento se puede, clavar, pegar o atornillar para hacer que la unión sea firme. Por otra parte, el método de cola de milano o cola de pato consiste en ranurar en sentidos contrarios los extremos de dos piezas a unir, es decir, donde una pieza tendrá una ranura hembra, la otra pieza tendrá un macho, que hará que estas embonen correctamente para impedir que se muevan. (Albano, 2012) En la Imagen 2.2 se ejemplifica el principio de dichas metodologías antes mencionadas.

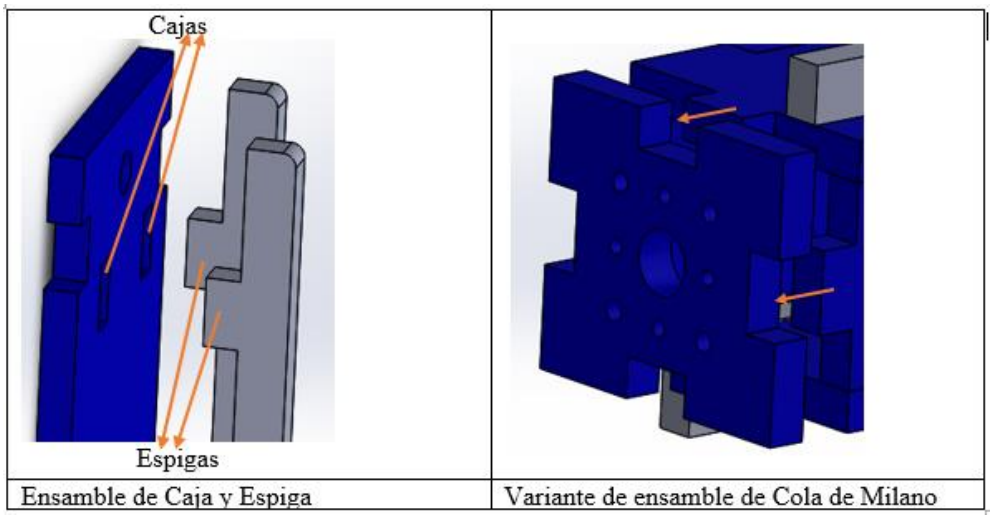


Imagen 2.2. Métodos de ensamble empleados en el robot adaptable.

Para realizar el diseño del robot adaptable, es necesario conocer los elementos principales sobre los cuales se basará su construcción, estos elementos fundamentales se pueden ubicar dentro de los siguientes sistemas:

- Mecánico
- Electrónico
- Programación

2.5 Selección del tipo de actuadores a utilizar

Actuadores

Definición: Son dispositivos inherentemente mecánicos capaces de transformar energía hidráulica, neumática o eléctrica con la finalidad de proporcionar fuerza para mover o actuar otro dispositivo mecánico.

Dependiendo del tipo de la fuente de energía con que se alimenta al actuador, estos pueden ser:

- Neumáticos
- Hidráulicos
- Eléctricos

En la elaboración del robot adaptable se ha decidido utilizar actuadores eléctricos, pues tienen una mayor facilidad de uso, mayor practicidad para ser alimentados con energía eléctrica y porque son de menor tamaño, pues se busca que el robot sea compacto.

2.5.1 Actuadores eléctricos

En aplicaciones de robótica se prefiere utilizar actuadores eléctricos en vez de cualquier otro tipo existente (neumático e hidráulico). Éstos presentan una gran facilidad para trabajar por sus características de control, fácil manejo y precisión, las características anteriores hacen que sean los más utilizados.

Para éste trabajo se emplearon motores de corriente directa, que se pueden clasificar de la siguiente manera:

- Motores de corriente directa:
  - Motor a pasos
  - Motorreductores
  - Servomotor

Imagen 2.3 Motorreductor.

Imagen 2.4 Servomotor.

Imagen 2.5. Motor a pasos 28BYJ-48.

2.5.2 Selección de motores

Se planteó la posibilidad de trabajar únicamente con motorreductores y servomotores, puesto que los motores a pasos no serían de gran utilidad para la aplicación requerida en el robot adaptable, ya que son de gran tamaño, generan poco par y son poco precisos, pues trabajan en intervalos de ángulos preestablecidos.

Para el robot adaptable se decidió utilizar Micro-motorreductores POLOLU (imagen 2.6), porque es un motorreductor pequeño y precisamente es por esta cualidad que lo hace apto para ser seleccionado como un motor a emplear, además es capaz de manejar diferentes pares, que van desde 1.41 [N.cm] hasta 88.27 [N.cm], según sea la reacción de engranaje que éste tenga.

Imagen 2.6 Micro motorreductor POLOLU.

Los servomotores que se decidieron utilizar para el robot adaptable fueron; los servomotores Dynamixel de la serie RX (imagen 2.7), porque son pequeños y capaces de soportar una carga que van desde 130 [N.cm] hasta los 530 [N.cm], además de que se puede obtener información de los mismos (se pueden leer valores de operación en el servomotor como: temperatura, posición, velocidad de desplazamiento, entre otros).

Imagen 2.7 Servomotor Dynamixel Rx-28.

2.6 Sensores

El uso de sensores en aplicaciones de robótica es de gran utilidad, pues estos nos brindan una mejor precisión al momento de seguir alguna trayectoria o cuando se busca alcanzar una posición deseada con mayor exactitud. Tal es el caso del robot adaptable, donde se busca que el uso de sensores nos dé un mejor manejo del mismo.

Lo sensores que se utilizaron en el robot adaptable son: encoders y micro-switches. Los micro-motorreductores POLOLU controlan el movimiento de las juntas prismáticas del

eslabón adaptable. Los encoders permitirían estimar la posición en la que se encuentra desplazada la longitud del eslabón del robot adaptable, Los Micro-Switches sirven como medida de seguridad, detectando las posiciones iniciales y finales asegurando que no saldrá de los límites de trabajo establecidos.

2.6.1 Micro-Switch

Los Switches o interruptores son, básicamente, dispositivos que permiten el paso o la interrupción del curso de una corriente eléctrica, con la finalidad de hacer que una señal sea detectada o no (Encendido/Apagado).

Para fines del robot adaptable, la aplicación principal de los micro-switches es que proporcionen una señal de encendido o apagado para que esto se traduzca, mediante programación, en una instrucción de paro (Límite superior o inferior del robot adaptable), pues fungen como botones de emergencia en caso de que no se detenga el motor en los límites establecidos de los eslabones adaptables. La imagen 2.8 nos muestra el tipo de micro-switches utilizados.

Imagen 2.8 Micro-Switch.

Características de los micro-Switch utilizados:

- Voltaje de operación: hasta 250 VAC.
- Corriente de operación: Hasta 5 A.
- Vida eléctrica: 10,000 ciclos como mínimo.

2.6.2 Encoders

Existen 2 tipos de encoders y estos pueden ser incrementales y absolutos, dentro de estos dos tipos, a su vez, también pueden ser ópticos, lineales y de cuadratura. El tipo de encoders empleados para establecer el estado de las juntas prismáticas son los encoders de cuadratura, ya que es el tipo de encoder que maneja de fábrica el Micro motorreductor POLOLU (imagen

2.9). Estos encoders son de tipo rotativo incremental y tiene la capacidad de indicar tanto la posición, la dirección y la velocidad del movimiento.

Características principales de los encoders:

- Encoders de tipo magnético.
- Alta resolución.
- Puede trabajar en un gran espectro de velocidades (rpm).
- Extremadamente duraderos.

Imagen 2.9 Encoder magnético y micro-motorreductor POLOLU.

2.7 Tarjeta MD25

La tarjeta MD25 (imagen 2.10) es un controlador dual de motores, diseñada para el uso de motores EMG30.Las características principales son:

1. Conteo de los encoders para determinar la distancia recorrida y la dirección.
2. Controla dos motores con un control independiente o combinado.
3. Se puede leer la corriente de cada motor.
4. Se requieren 12 V para alimentar el módulo.
5. Corriente de salida hasta 2.8 A para cada motor.
6. Los motores se pueden controlar la velocidad y dirección por el valor que se le mande.
7. Incluye control de aceleración.
8. Interface I2C que permite conectar hasta 8 drivers MD25 en el mismo módulo.

Imagen 2.10 Tarjeta MD25.

2.8 Microcontrolador

Un microcontrolador es un circuito integrado que en su interior tiene tres componentes principales (CPU, memoria y unidades de entrada y salida). Dicho microcontrolador es programable y su función principal es realizar una determinada tarea con un programa que el usuario grabe en su memoria interna.

Para este trabajo se utiliza un microcontrolador llamado Arduino MEGA 2560 (imagen 2.11), éste microcontrolador es fabricado por Arduino. Se decidió utilizar éste microcontrolador debido a que es una tarjeta de desarrollo de propósito general, fácil de programar, existen una gran cantidad de librerías que facilitan su uso, tiene bajo costo adquisitivo y es una tarjeta muy utilizado a nivel global

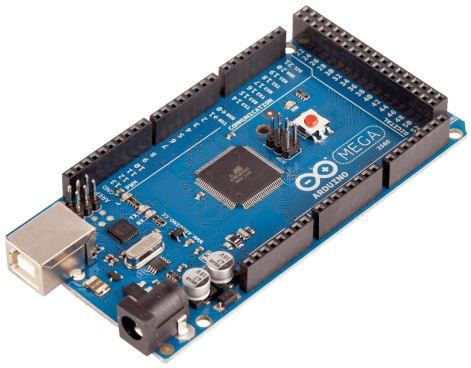


Imagen2.11 Arduino MEGA 2560.

Capítulo 3

Diseño a detalle y construcción

El robot adaptable fue diseñado de manera modular, es decir, que está conformado por varios elementos y si alguna de sus partes se descompone, se rompe o simplemente se desea cambiar por algún otro elemento, éste puede ser reemplazado con facilidad. En el caso de un eslabón, éste podría ser reemplazado con un eslabón adaptable o no adaptable, de igual forma es el caso de la base en la que se encuentra actualmente, ya que ésta es fija y sí en trabajos futuros éste robot adaptable pudiera estar en una plataforma móvil o incluso podría formar parte de un robot hibrido. Además, si se quisiera usar un efector final en el robot adaptable, bastaría con colocarle alguno existente en el mercado o alguno de fabricación personal, porque tiene la característica de ser modular.

Los módulos principales con los que cuenta el robot son dos: la base que soporta a todo el robot adaptable y los eslabones del robot que a su vez están diseñados como parte interior y parte exterior.

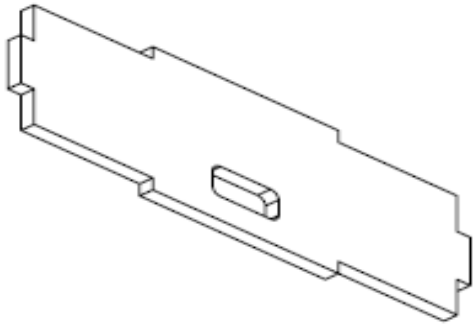
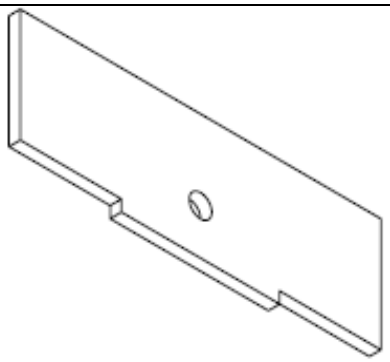
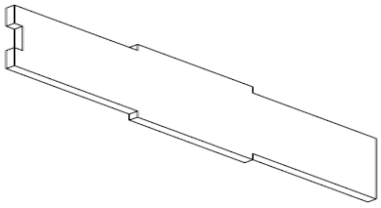
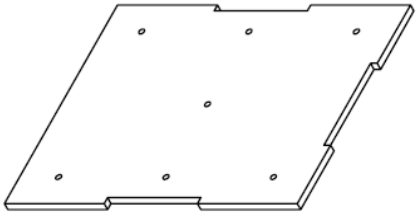
El diseño de las piezas fue hecho en un software de CAD y fueron pensadas para ser fabricadas en acrílico transparente, con espesor de 5 mm y ser cortadas mediante corte laser.

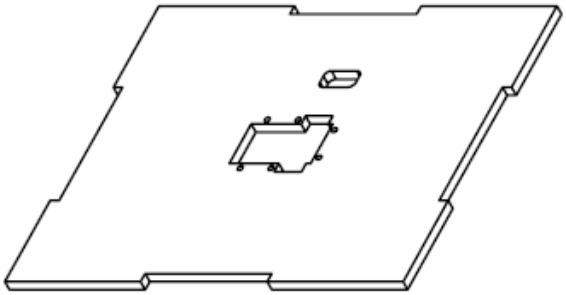
3.1 Base del robot adaptable

La base del robot consta de seis piezas, de las cuales sólo dos son exactamente iguales como se muestra en la tabla 3.1. Lo que se buscó con la base, fue tener un buen soporte donde montar al robot adaptable con la finalidad de evitar que éste se caiga.

Para el diseño de la base se planteó tener un espacio al interior, con la finalidad de guardar el microcontrolador Arduino, la tarjeta MD25, un módulo de comunicación TTL-RS485 y una pequeña placa diseñada con headers hembra, para tener las conexiones de 5 V y tierra que se requieren para la alimentación y correcto funcionamiento de los diversos circuitos integrados empleados.

Tabla 3.1 Descripción e imagen de las piezas de la base del robot adaptable.

Descripción	Imagen
N° de piezas: 1 Nombre CAD: Caja lados corto Esta pieza es la tapa posterior de la base del robot adaptable, tiene un orificio por donde salen los cables de alimentación de voltaje y de comunicación entre la computadora y las tarjetas MD25 y Arduino.	
N° de piezas: 1 Nombre CAD: Caja lados corto bisagra bueno Ésta pieza es la tapa frontal de la base del robot adaptable, sirve como puerta de acceso a los componentes en el interior de la base. El orificio que tiene es para colocar una pequeña agarradera con el fin de ayudar a tener un mejor agarre cuando se quiera abrir o cerrar.	
N° de piezas: 2 Nombre CAD: Caja lados largo Ésta pieza es una de dos piezas que van a los lados de la base del robot adaptable, sirven como paredes y soporte.	
N° de piezas: 1 Nombre CAD: Caja tapa abajo Ésta pieza es la tapa que va en la parte de abajo de la base del robot adaptable, tiene una serie de orificios, los cuales sirven para atornillar la base a una superficie con la finalidad de que ésta quede fija.	

N° de piezas: 1 Nombre CAD: Caja tapa arriba Ésta pieza es la tapa superior de la base del robot adaptable. El orificio más grande es por donde se inserta uno de los servomotores, el orificio más pequeño es para que salgan los cables de comunicación de los sensores, motores y servomotores con las tarjetas MD25 y Arduino.	

En las imágenes 3.1 y 3.2, se muestra una comparativa entre el modelo diseñado para la base del robot adaptable en el software de CAD, con respecto a la construcción final de la misma. La imagen 3.3 muestra las dimensiones finales de la base del robot adaptable.

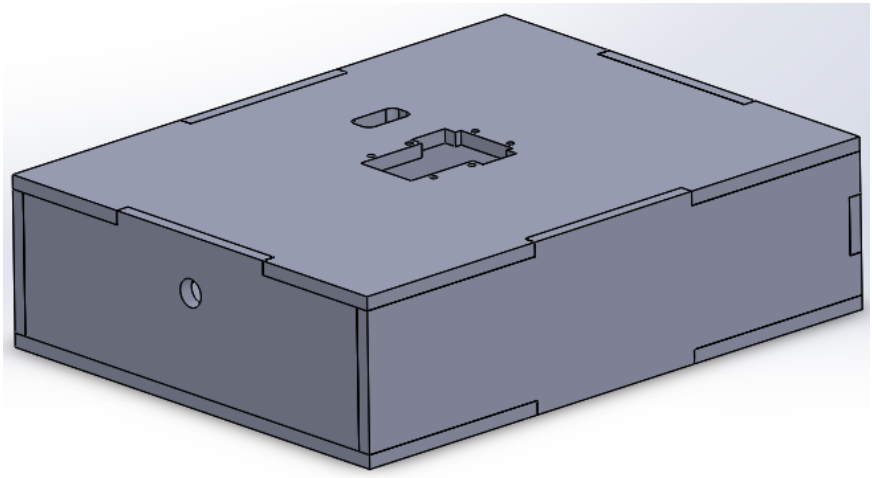


Imagen 3.1 Estructura de la base del robot adaptable diseñada en CAD.

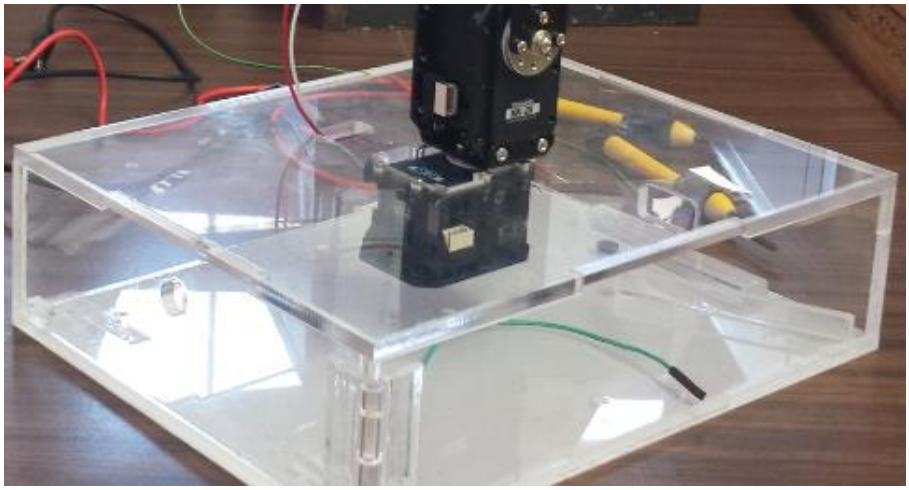


Imagen 3.2 Estructura de la base del robot adaptable construida.

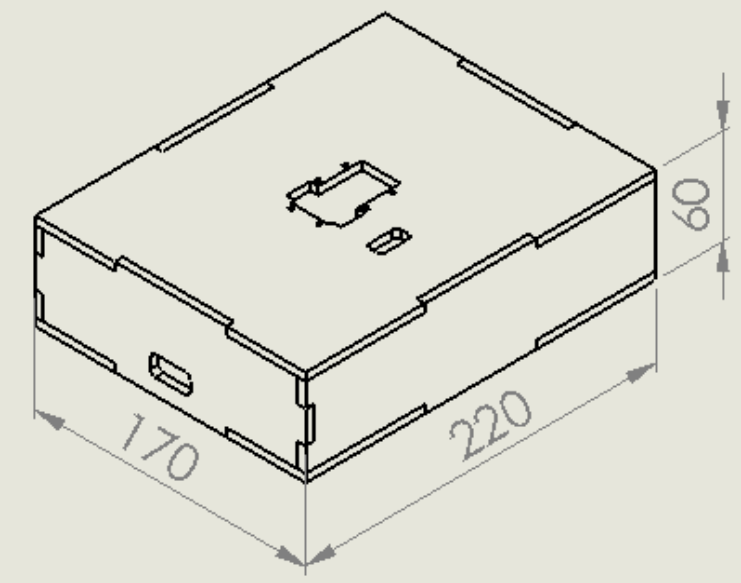


Imagen 3.3 Dimensiones de la base del robot adaptable en milímetros.

3.2 Eslabón adaptable

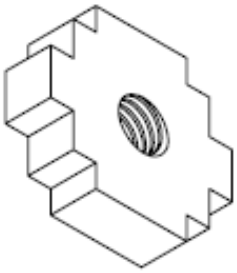
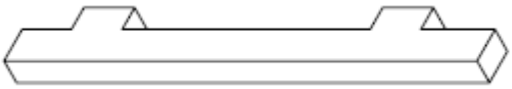
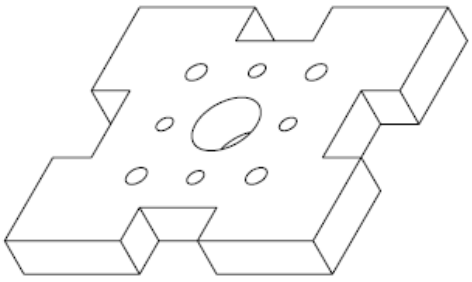
Como ya se mencionó anteriormente, el diseño del eslabón adaptable se dividió en dos partes, para su elaboración, en parte interior (parte móvil) y parte exterior (parte fija), ya que su diseño está basado en el principio de funcionamiento de los mecanismos empleados en las guías correderas para cajón. Se diseñó de esta manera porque se cuenta con una parte fija (parte exterior) y una parte con desplazamiento (parte interior). Su principio de operación se asemeja a la acción de abrir y cerrar un cajón, donde la función principal de las guías corredera es el permitir un mejor desplazamiento entre la base fija y la móvil, debido a que se minimiza la fricción existente, además le da un soporte firme por donde desplazarse, impidiendo que la parte móvil se desvié de fuera de los rieles guía. En la imagen 3.4 se muestran las guías correderas para cajón.

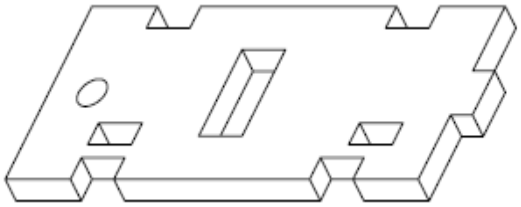
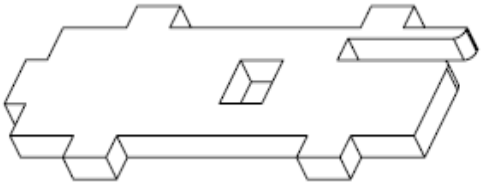
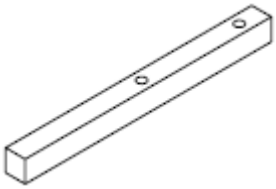
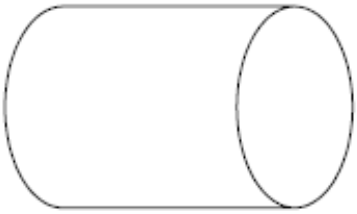
Imagen 3.4 Guías corredera para cajón con extracción parcial.

3.2.1 Eslabón interior

La parte interior del eslabón adaptable consta de nueve piezas de acrílico, una pieza fabricada mediante impresión 3D, dos rodamientos FUJI 686zz y tres Brackets para servomotores Dynamixel de la serie RX, denominados como: FR07-N101, FR07-i101 y FR07-S101.En la tabla 3.2 se detallan las características de los elementos antes mencionados.

Tabla 3.2 Piezas que forman parte del eslabón interior del robot adaptable.

Descripción	Imagen
N° de piezas: 1 Nombre CAD: Hembra tornillo Ésta pieza está sujeta a las paredes del eslabón interior, con la finalidad de que mediante la utilización del mecanismo de un tornillo sin fin y un micro-motorreductor Pololu, el eslabón interior pueda salir y entrar en el eslabón exterior (Desplazarse).	
N° de piezas: 2 Nombre CAD: Pestaña rodamiento Ésta pieza funge como una guía de desplazamiento para el eslabón interior, impidiendo que éste sufra alguna desviación en su trayectoria.	
N° de piezas: 1 Nombre CAD: Tapa interior La función principal de ésta pieza es: servir como soporte para un servomotor Dynamixel, con la finalidad de tener un robot adaptable modular serial, donde teóricamente se pueda colorar múltiples eslabones.	
N° de piezas: 2	

Nombre CAD: M soporte rodamiento Ésta pieza sirve como apoyo para la pieza “hembra tornillo” y además forma parte de las paredes del eslabón interior. Tiene un orificio circular por donde se coloca un perno junto con un rodamiento, en las ranuras rectangulares se empotran las pestañas de la guía del eslabón.	
N° de piezas: 2 Nombre CAD: M soporte rodamiento lados Ésta pieza forma parte de las paredes del eslabón interior y además soporta a la pieza “hembra tornillo”, tiene una ranura en la parte inferior derecha para evitar chocar con un soporte colocado en el eslabón exterior que ayuda a sujetar al micro-motorreductor Pololu.	
N° de piezas: 1 Nombre CAD: Poste m Switch Ésta pieza sirve como ayuda para presionar a los micro-switches y va colocada sobre la pieza “tapa interior”.	
N° de piezas: 2 Nombre CAD: Hembra tornillo Ésta pieza es donde los rodamientos se sujetan a las paredes del eslabón interior. Estas piezas no fueron fabricadas de una placa de acrílico, sino que fueron maquinadas de una varilla de acrílico.	

3.2.1.1 Rodamientos

Los rodamientos empleados en el eslabón del robot adaptable, fueron unos rodamientos rígidos de bolas 686zz (imagen 3.5). La utilización de éstos rodamientos es, principalmente, para implementar el mismo principio de funcionamiento del mecanismo que tienen las guías correderas para cajón, para que ayuden a disminuir la fricción existente entre las placas de acrílico y además dé soporte cuando el eslabón interior esté completamente extendido hacia afuera del eslabón exterior.

Tabla 3.3 Especificaciones del rodamiento 686zz.

Modelo	Rodamiento 686zz
Tipo	Rodamiento rígido de bolas
Marca	FUJI
Diámetro interior (d)	6 mm
Diámetro exterior (D)	13 mm
Espesor (B)	5 mm

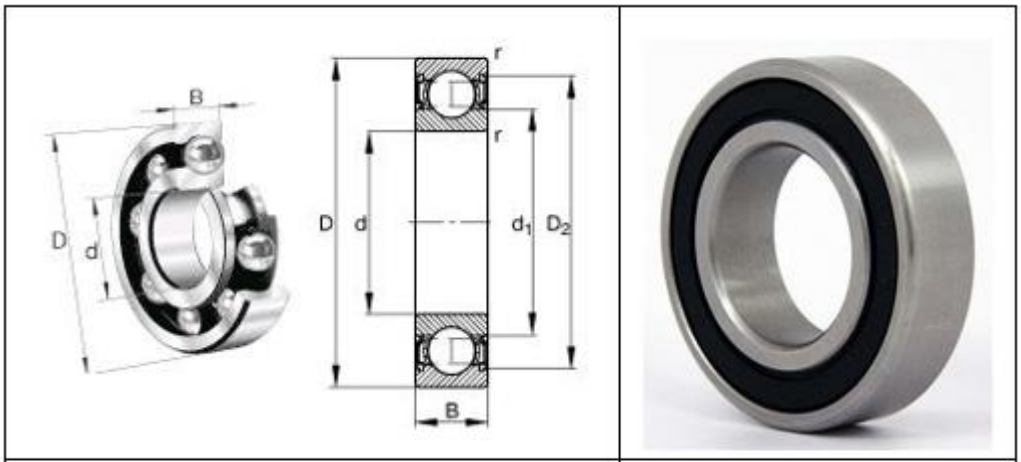


Imagen 3.5 Dimensiones del rodamiento FUJI 686z

3.2.1.2. Bridas FR07-N101 y FR07-i101

Las bridas FR07-N101 y FR07-i101, forman parte de la articulación rotacional que se encuentra en el eje de rotación del servomotor Dynamixel. Se empleó este tipo de bridas porque son las que ocupan los Dynamixel de la serie RX, como lo muestra la imagen 3.6,

mientras que en las imágenes 3.7 y 3.8 se muestran las dimensiones de las bridas FR07-N101 y FR07-i101 respectivamente. El material en el que están fabricadas las bridas es de aluminio, sin especificar (ROBOTIS, 2009a) (ROBOTIS, 2009b).

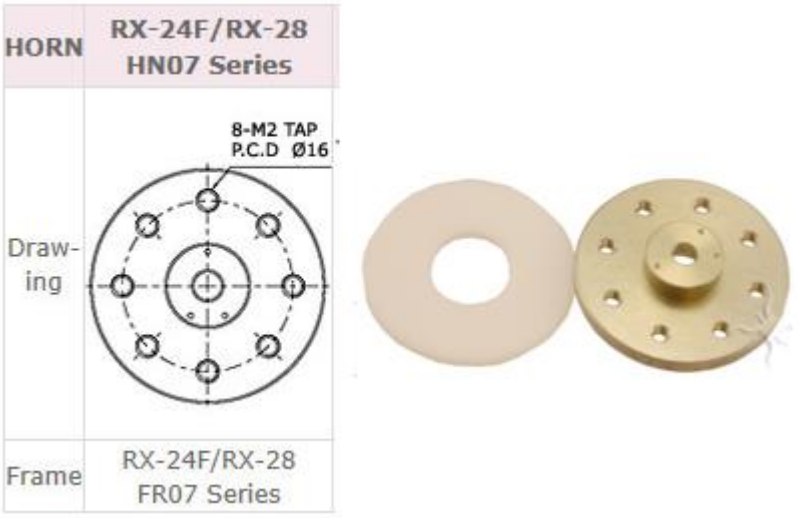


Imagen 3.6 Brida FR07-N101.

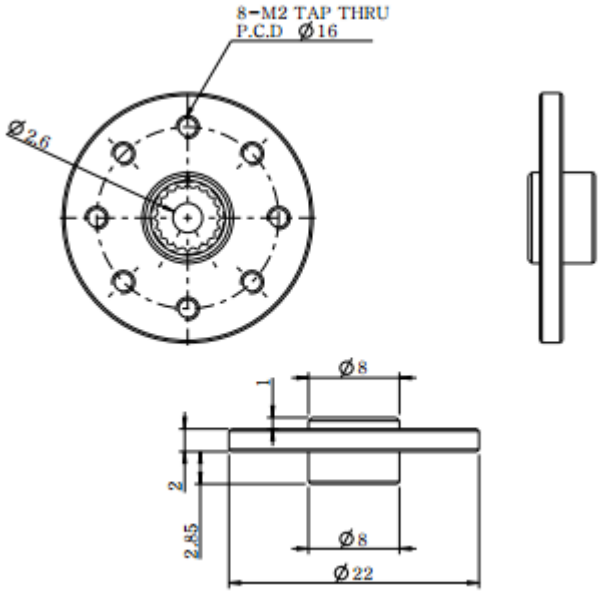


Imagen 3.7 Dimensiones de la brida FR07-N101.

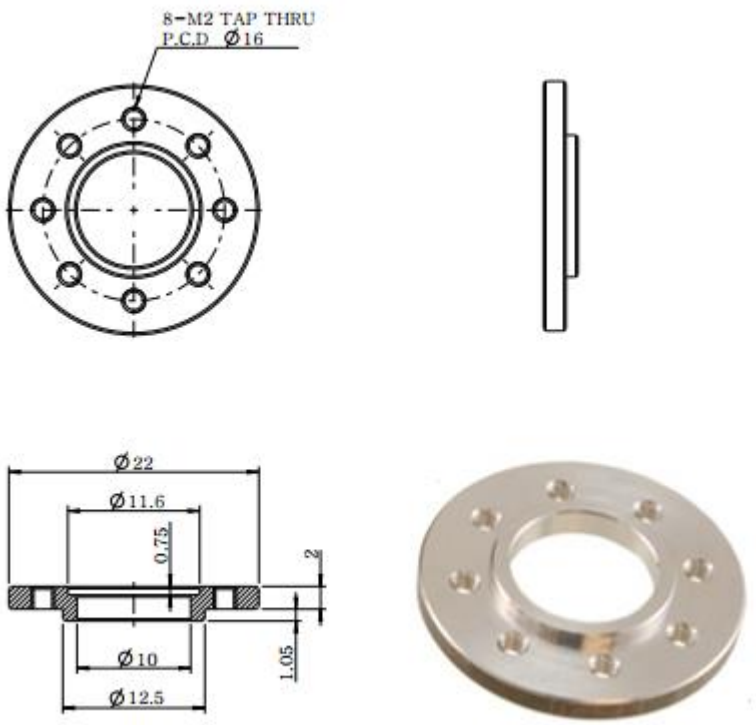


Imagen 3.8 Dimensiones de la brida i101.

3.2.1.3. Bracket FR07-S101

El propósito de utilizar el Bracket FR07–S101 es, poder tener una buena unión entre dos servomotores Dynamixel Rx 28 y/o poder unir uno de esos mismos servomotores a la parte interior del eslabón del robot adaptable que está hecha de acrílico. El Bracket es de aluminio (sin especificar) y por lo tanto nos garantiza que se tendrá una unión firme entre las piezas, además el aluminio tiene buenas propiedades mecánicas que para este diseño quedan bastante sobradas por el tipo de carga que deberá soportar. En las imágenes 3.9 y 3.10 se muestran las dimensiones y el Bracket real, respectivamente (ROBOTIS, 2009c).

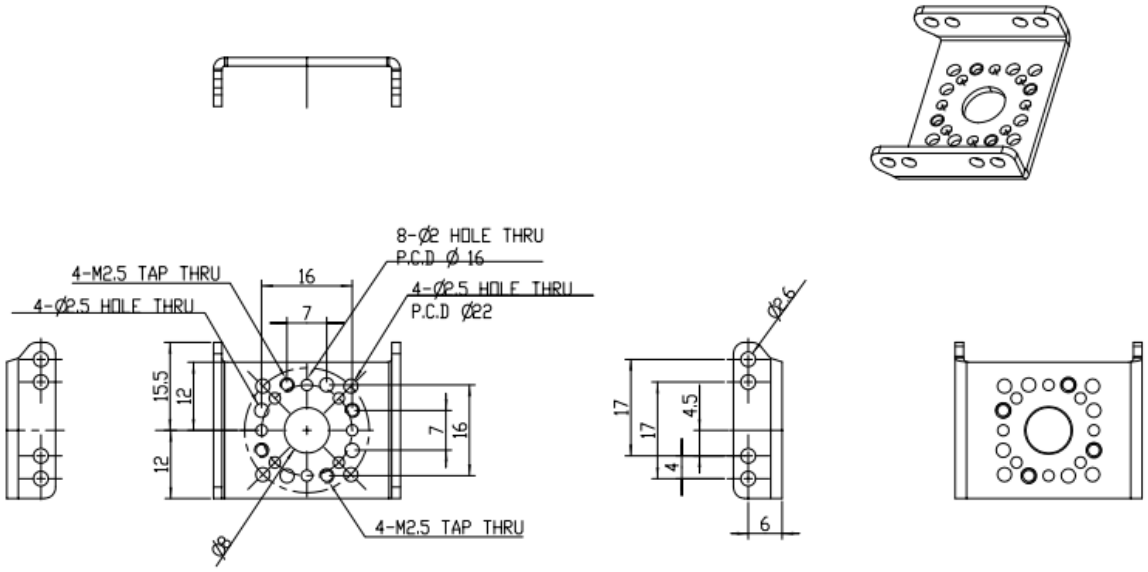


Imagen 3.9 Dimensiones del Bracket FR07-S101.

Imagen 3.10 Bracket FR07-S101.

3.2.1.4 Ensamble del eslabón interior

Una vez teniendo el diseño de las piezas de la parte interior del eslabón del robot adaptable, se realizó un ensamble para verificar que las piezas estuvieran correctamente, antes de enviarlas a cortar en laser. En la imagen 3.11, se muestra el diseño de la estructura del eslabón interior del robot adaptable, dibujada en CAD.

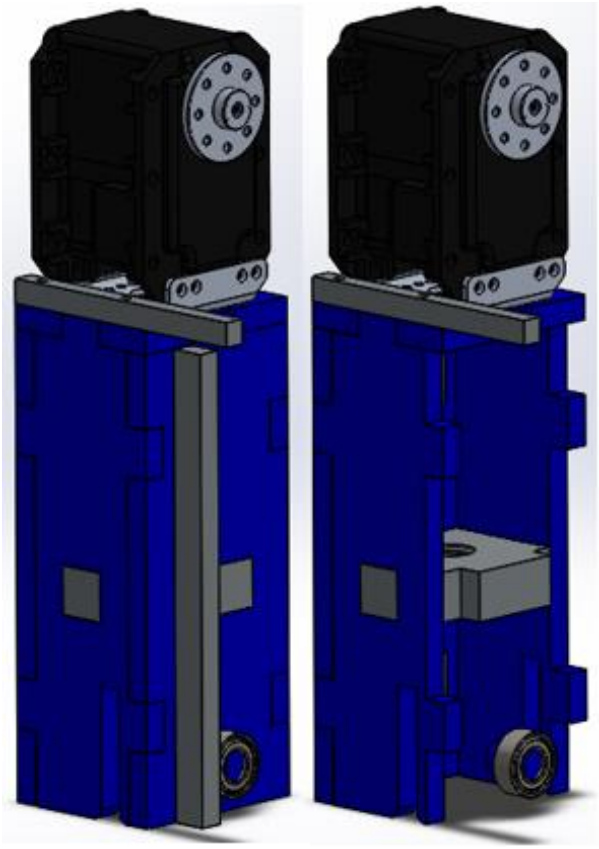
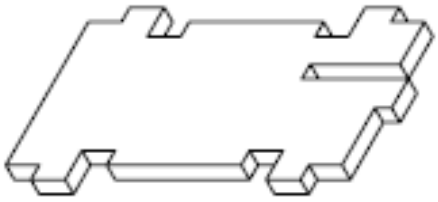
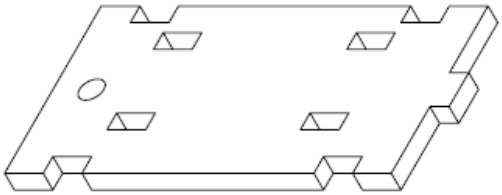
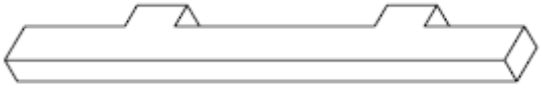


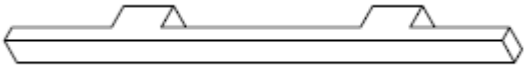
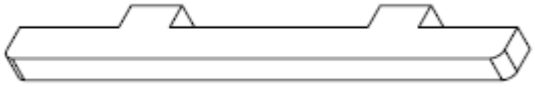
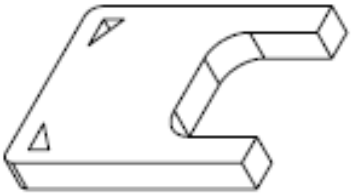
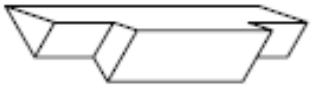
Imagen 3.11 Estructura del eslabón interior diseñada en CAD.

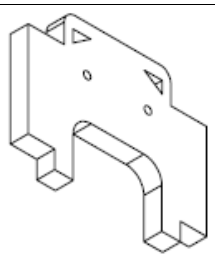
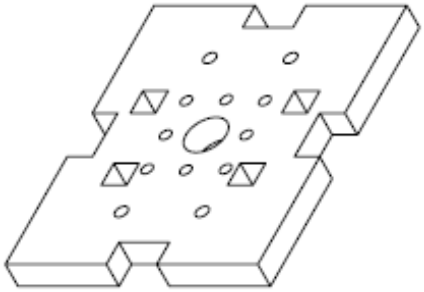
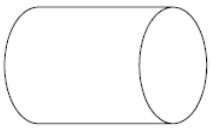
3.2.2 Eslabón exterior

El diseño para la parte exterior del eslabón del robot adaptable, está conformado por diecinueve piezas de acrílico, una pieza fabricada mediante impresión 3D, un Bracket FR07-H101 y dos rodamientos FUJI 686zz, además la parte exterior del eslabón del robot adaptable, tiene en su interior un micro-motorreductor Pololu y un soporte para el mismo que se encuentra atornillado en una base interna. En la tabla 3.4 se detallan las características de los elementos antes mencionados.

Tabla 3.4 Piezas que forman parte del eslabón exterior del robot adaptable.

Descripción	Imagen
N° de piezas: 2 Nombre CAD: Exterior tapa Ésta pieza forma parte de una de las paredes exteriores del eslabón adaptable, tiene una ranura en la parte inferior derecha, donde se empotra la pieza: "soporte para base Pololu", las demás ranuras son para insertar las pestañas de los rieles por donde se desplazan los rodamientos.	
N° de piezas: 2 Nombre CAD: Exterior rod tapa Al igual que la pieza anterior, ésta pieza forma parte de las paredes exteriores del eslabón del robot adaptable, con la diferencia de que en la ranura circular lleva un perno donde se sujeta uno de los rodamientos y en las ranuras rectangulares se insertan unas pestañas de los rieles guía del eslabón adaptable.	
N° de piezas: 2 Nombre CAD: Exterior lados riel Ésta pieza va unida a la pieza exterior tapa, con la finalidad de hacer que la pieza "ayuda soporte interior" no se mueva de la posición en la que está, ya que ésta es importante dentro del diseño, porque impide que el eslabón se mueva fuera de esa guía.	
N° de piezas: 2	

Nombre CAD: Exterior lados riel chico Al igual que la pieza anteriormente descrita, la función principal de ésta pieza es; evitar que la pieza “ayuda soporte interior” se mueva.	
N° de piezas: 4 Nombre CAD: Ayuda soporte interior Las pestañas de ésta pieza, van insertadas en las ranuras de la pieza “exterior rod tapa”. Su función principal es evitar que el eslabón se mueva fuera de las guías y además evita que se tambalee cuando la parte interior del eslabón se encuentra completamente extendido.	
N° de piezas: 1 Nombre CAD: Soporte Pololu 2 Ésta pieza ayuda a evitar que el micro-motorreductor Pololu se mueva, ya que está unida a las piezas “poste soporte Pololu” y “soporte para base Pololu”.	
N° de piezas: 2 Nombre CAD: Poste soporte Pololu Ésta pieza es un poste que va insertado entre las piezas “Soporte Pololu 2” y “Soporte para base Pololu”, tiene una forma triangular para facilitar la instalación del micro-motorreductor Pololu.	
N° de piezas: 1 Nombre CAD: Soporte para base Pololu Ésta pieza es la base principal de donde se sujeta el micro-motorreductor Pololu, los	

orificios circulares son para atornillar la base sobre la cual va montado.	
N° de piezas: 1 Nombre CAD: Tapa exterior En los orificios cuadrangulares de esta pieza, se insertan las pestañas de las piezas “Soporte Pololu 2” y “Soporte para base Pololu”, además en los orificios circulares se atornilla el eslabón completo al bracket FR07-H101 que va en la parte giratoria del servomotor Dynamixel Rx 28.	
N° de piezas: 1 Nombre CAD: Tornillo ensamble Éste tornillo trabaja en conjunto con el micro-motorreductor Pololu y la pieza “hembra tornillo”, permitiendo que el eslabón interior tenga movimiento, el tornillo sin fin fue diseñado para ser fabricado en impresión 3D, es un tornillo M12.	
N° de piezas: 2 Nombre CAD: Hembra tornillo En ésta pieza es donde los rodamientos se sujetan a las paredes del eslabón interior. Estas piezas no fueron fabricadas de una placa de acrílico sino de una varilla de acrílico.	
N° de piezas: 1	

Nombre CAD: Soporte micro Switch Ésta pieza sirve como soporte para los micro-switches del robot adaptable y va sujeta a la pieza “exterior rod tapa”.	

3.2.2.1 Micro-motorreductor POLOLU

Cada eslabón del robot adaptable cuenta con un micro-motorreductor POLOLU y un soporte para el mismo, tal y como se muestran en las imágenes 3.12 y 3.13, respectivamente. Cada micro-motorreductor tiene integrado un encoder magnético, como el que se muestra en la Imagen 3.12.

Imagen 3.12 Micro-motorreductor POLOLU con encoder magnético.

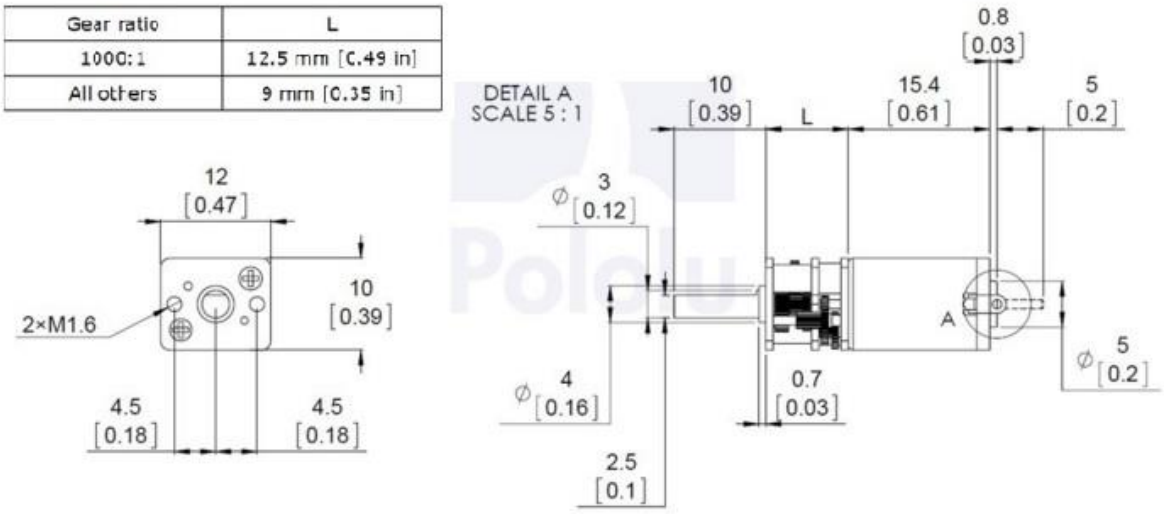


Imagen 3.13 Dimensiones reales del micro-motorreductor POLOLU.

3.2.2.2 Soporte Micro-motorreductor POLOLU

Los soportes utilizados para sujetar a los micro-motorreductores son los mostrados en la imagen 3.14. Estos soportes son los que vende el fabricante, específicamente para ser utilizados con los micro-motorreductores POLOLU. En la Imagen 3.15 se pueden apreciar las medidas de éste soporte.

Imagen 3.14 Soporte para micro-motorreductor POLOLU.

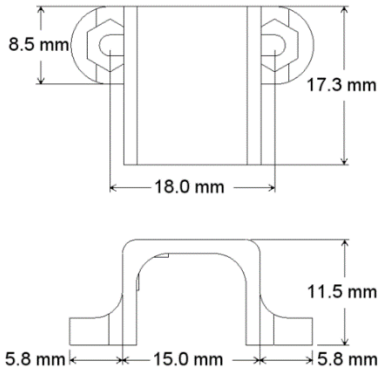


Imagen 3.15 Dimensiones reales del soporte para el micro-motorreductor POLOLU.

3.2.2.3 Encoder para el micro-motorreductor POLOLU

En la imagen 3.16 se muestran las medidas reales que tiene el encoder del micro-motorreductor POLOLU y en la Imagen 3.17 se puede apreciar el encoder y sus pines de conexión.

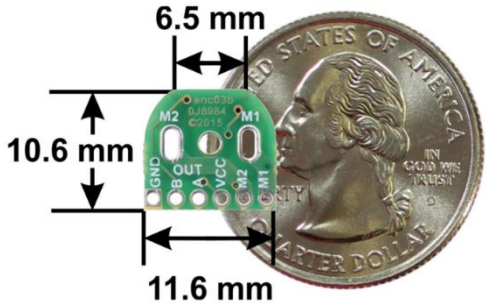


Imagen 3.16 Dimensiones reales del encoder magnético.

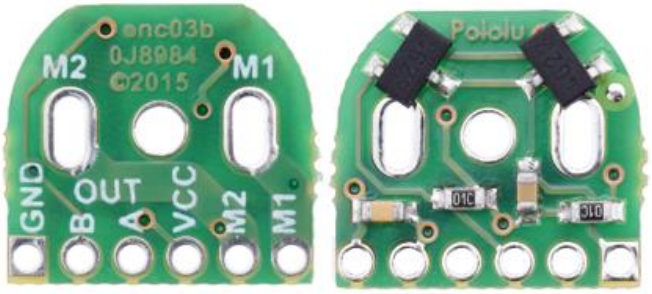


Imagen 3.17 Vista frontal y trasera del encoder magnético.

3.2.2.4 Diagrama esquemático del encoder

En la imagen 3.18 se puede apreciar el diagrama de conexión entre el micro-motorreductor POLOLU y el encoder magnético. El encoder, en su interior, cuenta con dos sensores de efecto Hall TLE4946-2K (U1 y U2), ambos visibles y los podemos apreciar como dos rectángulos negros y pequeños en la imagen 3.18.

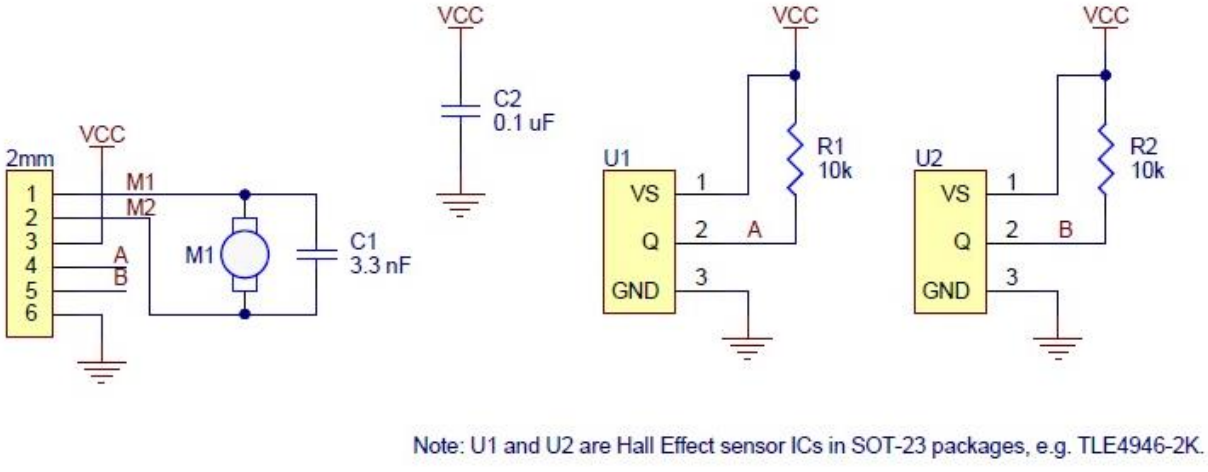


Imagen 3.18 Diagrama esquemático del encoder magnético.

3.2.1.2 Bracket FR07-H101

En el diseño del robot adaptable, se contempló utilizar el juego de Brackets compatibles con la serie Rx de los servomotores Dynamixel utilizados. El uso del Bracket FR07-H101 facilita el ensamble entre el servomotor y la tapa de acrílico sobre la cual va montada la estructura del eslabón adaptable. En las imágenes 3.19 y 3.20 se muestran las dimensiones del Bracket, así como el Bracket real, respectivamente (ROBOTIS, 2009d).

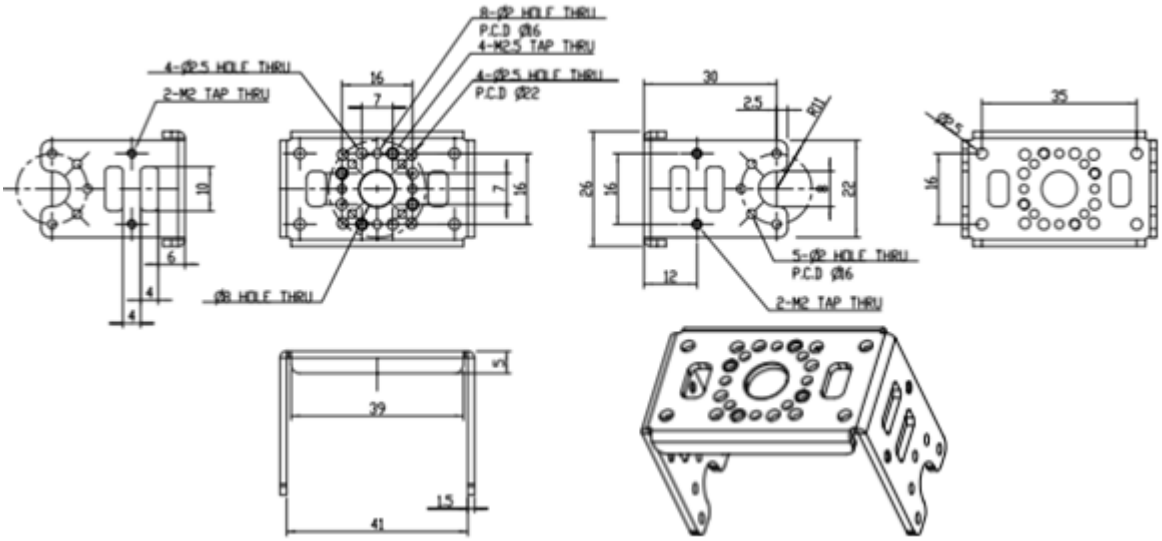


Imagen 3.19 Dimensiones del Bracket FR07-H101.

Imagen 3.20 Bracket FR07-h101.

3.2.2.5 Estructura del eslabón exterior

Con la finalidad de corroborar que el diseño y las piezas estuvieran correctas, se realizó un ensamble con todos los elementos presentes en la parte exterior del eslabón del robot adaptable. En la imagen 3.21, se muestra el ensamble de la estructura en CAD, así como los componentes electrónicos, antes mencionados.

Imagen 3.21 Estructura del eslabón exterior diseñada en CAD.

3.2.3 Eslabón adaptable completo

Una vez terminado los ensambles de las estructuras de las partes interior y exterior, se realizó el ensamble completo de toda la estructura del eslabón adaptable, para verificar que el diseño estuviera correcto y no hubiera alguna interferencia mecánica entre las piezas y que tuviera la movilidad esperada. En las imágenes 3.22 y 3.23 se muestra el ensamble de la estructura completa del eslabón adaptable contraída y extendida, respectivamente. Se puede ver como interacciona la parte interior y exterior del eslabón adaptable.

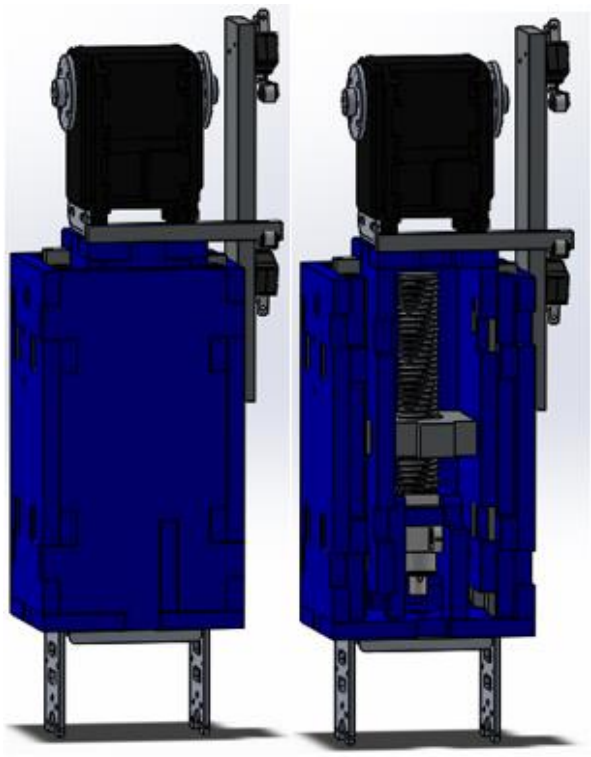


Imagen 3.22 Estructura del eslabón adaptable completa (contraído), diseñada en CAD.

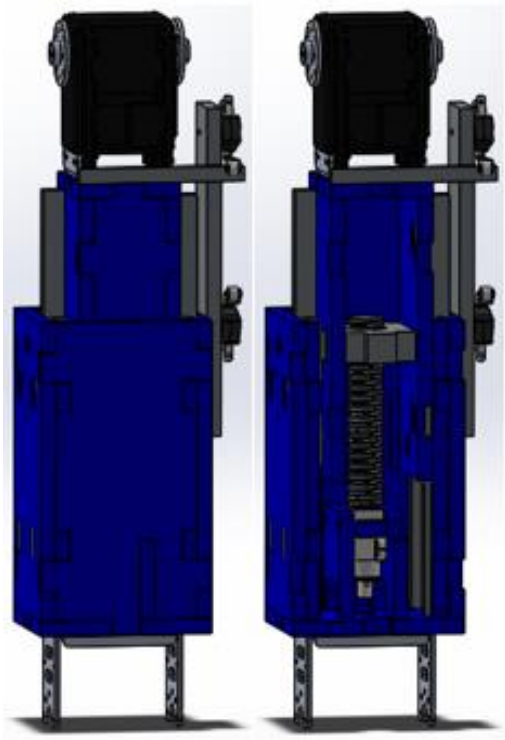


Imagen 3.23 Estructura del eslabón adaptable completa (extendido), diseñada en CAD.

Imagen 3.24 Primeras pruebas de la estructura del eslabón adaptable real.

3.3 Robot adaptable

Finalmente, como resultado de realizar las iteraciones anteriormente descritas en el presente capítulo, se ha llegado a la elaboración del prototipo de robot adaptable. En la imagen 3.25 se muestra el prototipo final del robot adaptable y el ensamble de la estructura del mismo diseñado en CAD.

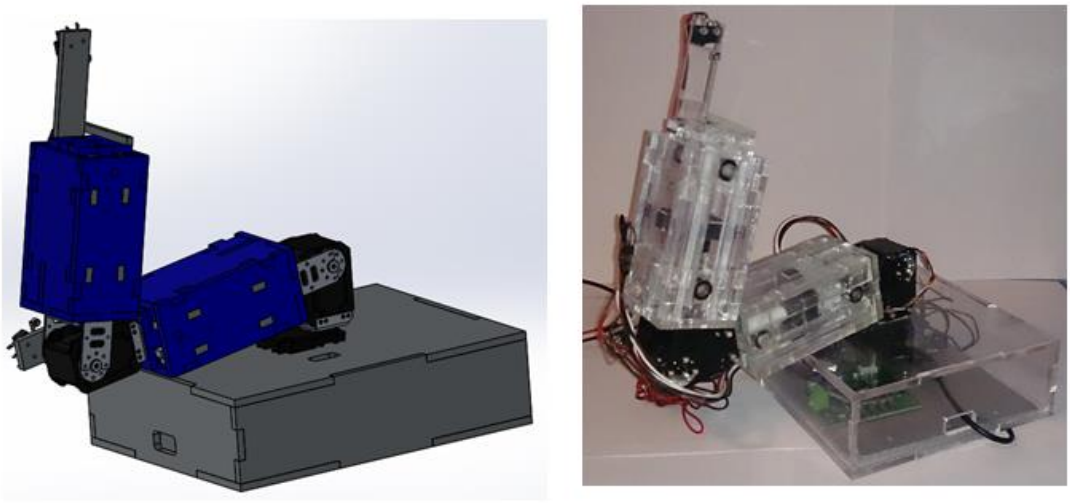


Imagen 3.25 Ensamble del prototipo en CAD y del prototipo real de un robot adaptable.

3.3.1 Análisis estático de cargas del robot adaptable

El realizar un análisis de cargas presentes en el robot adaptable, es una parte importante en el diseño del mismo, ya que con ello es posible saber si los servomotores empleados (Dynamixel Rx28 y Rx24) son capaces de soportar dichas cargas.

Para hacer el análisis de cargas es necesario tomar en cuenta las dimensiones de los eslabones, los elementos presentes en cada eslabón y el material del que están hechos. Con lo anterior, se utilizó un software de CAD para calcular las masas de los eslabones, asignando las propiedades de los materiales para cada uno de los elementos empleados, y así mismo el programa de CAD nos proporciona las coordenadas del centro de masa para cada eslabón.

En las imágenes 3.26 y 3.27, se muestran los diagramas de momentos que experimenta el robot adaptable y más en específico el servomotor 2, ya que es el que debe soportar una carga mayor en dos diferentes estados del robot adaptable, cuando éste se encuentra contraído (imagen 3.26) o extendido (imagen 3.27).

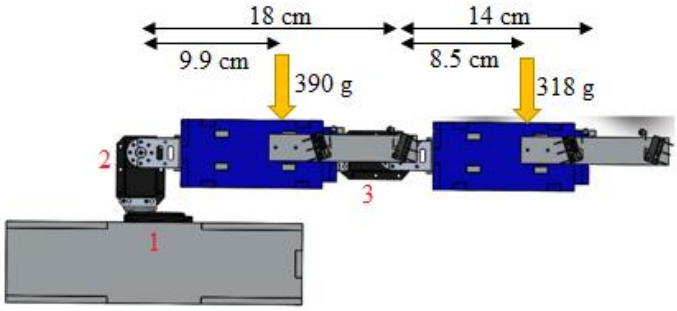


Imagen 3.26 Diagrama de momentos para el robot adaptable contraído.

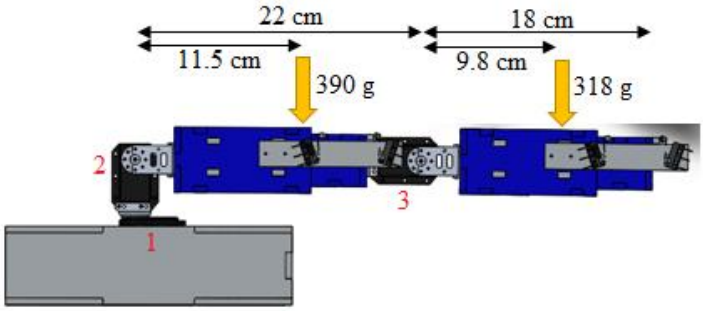


Imagen 3.27 Diagrama de momentos para el robot adaptable extendido.

El cálculo de los momentos se hace respecto del servomotor 2, en cada una de los diagramas (Imágenes 3.26 y 3.27), pues es donde se genera mayor brazo de palanca. Considerando los datos de las Imágenes 3.26 y 3.27 es posible realizar el siguiente cálculo:

$$M_{2\,Contraído} = (9.9\ cm)(0.390\ Kg) + (18\ cm + 8.5\ cm\ )(0.318\ Kg)$$
$$= \quad 12.288\ Kg \cdot cm$$

$$M_{2\,Extendido} = (11.5\ cm)(0.390\ Kg) + (22\ cm + 9.8\ cm\ )(0.318\ Kg)$$
$$= 14.597\ Kg \cdot cm$$

De los valores obtenidos para $M_{2\ \text{contraído}}$ y $M_{2\ \text{Extendido}}$ se puede decir que el servomotor 2, soportará sin problemas la carga en su punto más crítico, pues es un servomotor Dinamixel Rx28, que es capaz de soportar hasta una carga de 37 $Kg \cdot cm$.

3.3.2 Lista de componentes y materiales

En la tabla 3.4.1 se muestra una lista de todas las piezas, materiales y componentes electrónicos que se utilizaron para hacer al robot adaptable.

Tabla 3.4.1 Lista de componentes y materiales que conforman al robot adaptable

Cantidad	Componente	Descripción
Base del robot adaptable		
6	Piezas de acrílico trasparente de 5mm	Piezas que forman a la base del robot adaptable (Capítulo 3).
1	Servomotor Dynamixel Rx24	Hace girar al brazo del robot adaptable con respecto a la base.
1	Servomotor Dynamixel Rx28	Forma parte de una articulación dentro de un eslabón del robot adaptable.
2	Bridas FR07-N101	Van en el eje de giro de los servomotores, para ser acoplados a los Brackets FR07-S101.
1	Brida FR07-i101	Va sujeta junto con un rodamiento a un eje fijo en los servomotores Dynamixel y a su vez van acoplados a los Brackets FR07-S101.
1	Arduino Mega 2560	Tarjeta que se usa para la comunicación SIMULINK-Arduino-servomotores-Tarjeta MD25 (imagen 3.26).
1	Tarjeta MD25	Tarjeta controladora de los micro-motorreductores Pololu.
1	Módulo de comunicación RS-485	Modulo que convierte de TTL a RS-485.
1	Bracket FR07-S101	Va sujeto a la parte baja de un servomotor y en la tapa interior del eslabón interior (imagen 3.25)
1	Rodamiento MF106ZZ	Va sujeto al eje fijo del servomotor Dynamixel para evitar friccion entre el eje y el Bracket FR07-S101.
6	Tornillos de 2mm x 10mm	

6	Tuercas para tornillos de 2mm	Para unir un servomotor Dynamixel con la base del robot adaptable.
4	Tornillos 2.5mm x 6mm	Para sujetar un servomotor Dynamixel al Bracket FR07-S101.
4	Tuercas tuercas para tornillos de 2.5mm	
4	Tornillos de 2mm x 3mm	Para sujetar el Bracket FR07-S101 con la brida FR07-N101.
4	Tuercas para tornillos de 2mm	
1	Bisagra de acrílico de 1”	Sujeta una pieza de acrílico que funge como compuerta.
Primer eslabón del robot adaptable		
28	Piezas de acrílico trasparente de 5mm	Piezas que conforman al primer eslabón del robot adaptable. (Capítulo 3)
4	Rodamientos FUJI 686zz	Van en la parte exterior del robot interior (imagen 3.11) y al interior del robot exterior (imagen 3.21), para evitar la fricción.
1	Rodamiento MF106ZZ	Va sujeto al eje fijo del servomotor Dynamixel para evitar friccion entre el eje y el Bracket FR07-S101.
1	Tornillo sin fin impreso en 3D de ABS	Sirve para desplazar a la parte interior del eslabón adaptable (Imagen 3.23).
1	Piezas impresas en 3D de ABS	Soporte por donde gira un tornillo sin fin (imagen 3.23)
1	Brida FR07-N101	Van en el eje de giro de los servomotores, para ser acoplados a los Brackets FR07-S101.
1	Brida FR07-i101	Va sujeta junto con un rodamiento a un eje fijo en los servomotores Dynamixel y a su vez van acoplados a los Brackets FR07-S101.
1	Servomotor Dynamixel Rx28	Forma parte de una articulación dentro de un eslabón del robot adaptable.
1	Bracket FR07-H101	Conecta al eslabón adaptable con un servomotor Dynamixel (imagen 3.25)
1	Micromotorreductor Pololu	Hace girar al tornillo sin fin.
1	Soporte para Micromotorreductor Pololu	Para sujetar al Micro-motorreductor Pololu al interior de la parte exterior del robot adaptable (imagen 3.23)
2	Micro Switches con rodillo (Finales de carrera)	Delimitan la carrera de la parte interior del eslabón adaptable (imagen 3.25)
6	Tornillos de 2mm x 3mm	Para sujetar el Bracket FR07-H101 a las bridas FR07-N101 y FR07-i101.
4	Tornillos 2.5mm x 6mm	Para sujetar un servomotor Dynamixel al Bracket FR07-S101.
4	Tuercas tuercas para tornillos de 2.5mm	
4	Tornillos de 2mm x 10mm	

4	Tuercas para tornillos de 2mm	Para sujetar el Bracket FR07-S101 a la parte interior del robot adaptable (3.25).
4	Tornillos de 2mm x 10mm	Para sujetar el Bracket FR07-H101 a la parte exterior del robot adaptable (imagen 3.25)
4	Tuercas para tornillos de 2mm	
2	Pijas de 2mm x 10 mm	Para sujetar el soporte de los micro switches (imagen 3.25)
Segundo eslabón del robot adaptable		
28	Piezas de acrílico trasparente de 5mm	Piezas que conforman al primer eslabón del robot adaptable. (Capítulo 3)
4	Rodamientos FUJI 686zz	Va sujeto al eje fijo del servomotor Dynamixel para evitar friccion entre el eje y el Bracket FR07-S101.
1	Tornillo sin fin impreso en 3D de ABS	Sirve para desplazar a la parte interior del eslabón adaptable (Imagen 3.23).
1	Piezas impresas en 3D de ABS	Soporte por donde gira un tornillo sin fin (imagen 3.23)
1	Bracket FR07-H101	Conecta al eslabón adaptable con un servomotor Dynamixel (imagen 3.25)
1	Micro-motorreductor Pololu	Hace girar al tornillo sin fin.
1	Soporte para Micro-motorreductor Pololu	Para sujetar al Micro-motorreductor Pololu al interior de la parte exterior del robot adaptable (imagen 3.23)
2	Micro Switches con rodillo (Finales de carrera)	Delimitan la carrera de la parte interior del eslabón adaptable (imagen 3.25)
6	Tornillos de 2mm x 3mm	Para sujetar el Bracket FR07-H101 a las bridas FR07-N101 y FR07-i101.
4	Tornillos de 2mm x 10mm	Para sujetar el Bracket FR07-H101 a la parte exterior del robot adaptable (imagen 3.25)
4	Tuercas para tornillos de 2mm	
2	Pijas de 2mm x 10mm	Para sujetar el soporte de los micro switches (imagen 3.25)

3.4 Diseño electrónico

En éste apartado se definirán los elementos básicos electrónicos, indispensables para la correcta operación y funcionamiento del robot adaptable.

3.4.1 Microcontrolador Arduino Mega 2560

Las características generales del microcontrolador ATmega2560 (Arduino, 2016a).

- Trabaja con un voltaje de operación de 5V.
- Cuenta con 54 Pines digitales de entrada y salida, de los cuales 15 son salidas pwm.

- Cuenta con 16 Pines analógicos de entrada.
- En cada Pin de entrada y salida proporciona una corriente DC de 40mA.
- Cuenta con una memoria Flash con capacidad de 256 KB (8KB usados por el bootloader).
- SRAM: 8KB.
- EEPROM: 4KB.
- Clock speed: 16 MHz.
- Protocolos de comunicación: Serial, I^2C.
- Interrupciones internas: 6.

De las características más importantes que tiene el microcontrolador Arduino Mega 2560 sólo se utilizan los pines de comunicación serial, los de I^2C y los de interrupciones externas.

Los pines que se usan en el Arduino MEGA para la comunicación Serial se describen en la Tabla 3.5.

Tabla 3.5 Pines para Comunicación Serial Arduino MEGA.

	PIN RX	PIN TX
Serial	0	1
Serial 1	19	18
Serial 2	17	16
Serial 3	15	14

Los pines que se utilizan en el Arduino Mega para la comunicación I^2C son 20 (SDA) and 21 (SCL) y se requiere el uso de la librería “Wire”.

El Arduino Mega dispone de 6 pines que pueden ser configurados como interrupciones externas y se describen en la Tabla 3.6:

Tabla 3.6 Pines de Interrupciones externas Arduino MEGA.

Interrupción	PIN
0	2
1	3
2	21
3	20
4	19
5	18

Las interrupciones externas pueden ser activadas de 4 modos (LOW, CHANGE, RISING, FALLING)(Arduino, 2016b). A continuación, se describe el funcionamiento de cada una de ellas:

- LOW: Provoca la interrupción cada vez que al PIN le llega una señal baja.
- CHANGE: Provoca la interrupción cada vez que el PIN cambia de valor.
- RISING: Provoca la interrupción cuando el PIN va de un valor bajo a alto.
- FALLING. Provoca la interrupción cuando el PIN va de un valor alto a bajo

3.4.1.1 Uso del microcontrolador

Para el banco de pruebas del robot adaptable el microcontrolador Arduino MEGA se usa principalmente para establecer comunicación con los servomotores Dynamixel y la tarjeta MD25 se usa como una tarjeta de adquisición y procesamiento de datos, los cuales se obtendrán de SIMULINK (imagen 3.28).

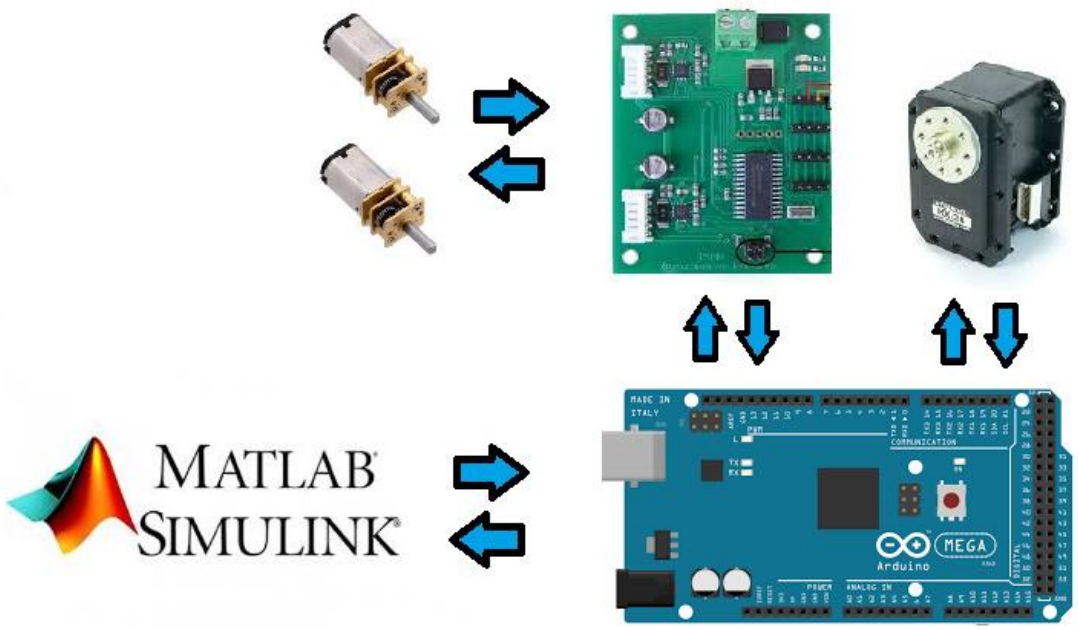


Imagen 3.28 Uso del microcontrolador Arduino MEGA.

3.4.2 Conexión Arduino - Dynamixel

Para establecer comunicación entre Arduino y los servomotores Dynamixel se utilizó la librería que proporciona Josué Alejandro Savage (S. Electronics, 2011). Esta librería se realizó de forma genérica para los servos Dynamixel, ya que se pueden utilizar las series AX, MX, RX, etc.

Las principales diferencias entre las distintas series de servos Dynamixel se pueden apreciar en la Tabla 3.7:

Tabla 3.7 Principales diferencias entre series de Dynamixel.

Series	Comunicación	Rango de movimiento
AX	TTL	300°
MX	TTL / RS-485	360°
RX	RS-485	300°
EX	RS-485	251°

Los conectores de los Dynamixel cambian dependiendo el protocolo de comunicación, para la comunicación TTL el conector tiene 3 PINES (VCC, GND y Data) y para la comunicación RS485 los conectores son de 4 PINES (VCC, GND, Data+ y Data-) (imagen 3.29).

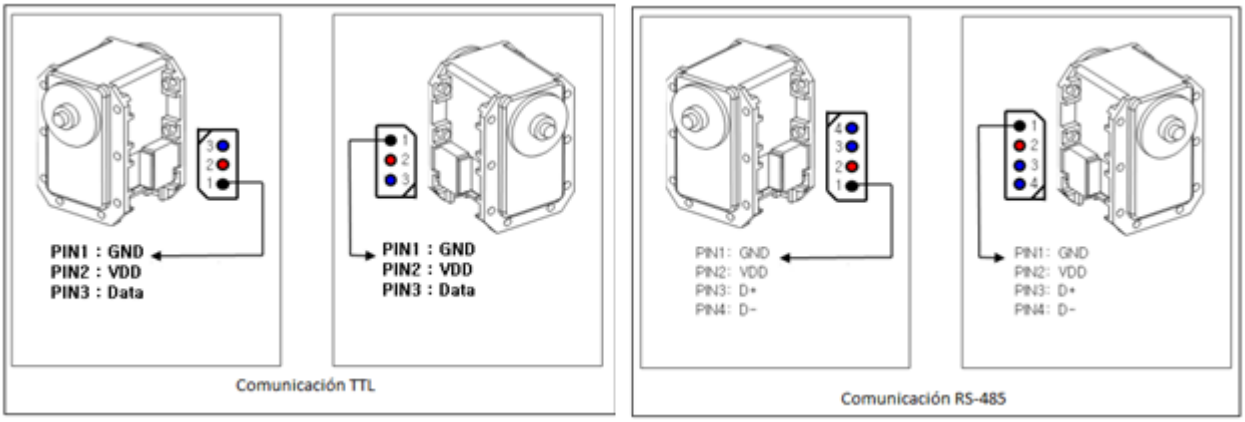


Imagen 3.29 Conectores Dynamixel RS-485 / TTL.

Para que ésta librería funcione con ambos protocolos de comunicación se requieren de dos integrados dependiendo el protocolo de comunicación. Para la comunicación TTL se requiere un 74LS241 y para la comunicación RS-485 se requiere un MAX 485 como lo muestra la Imagen 3.30.

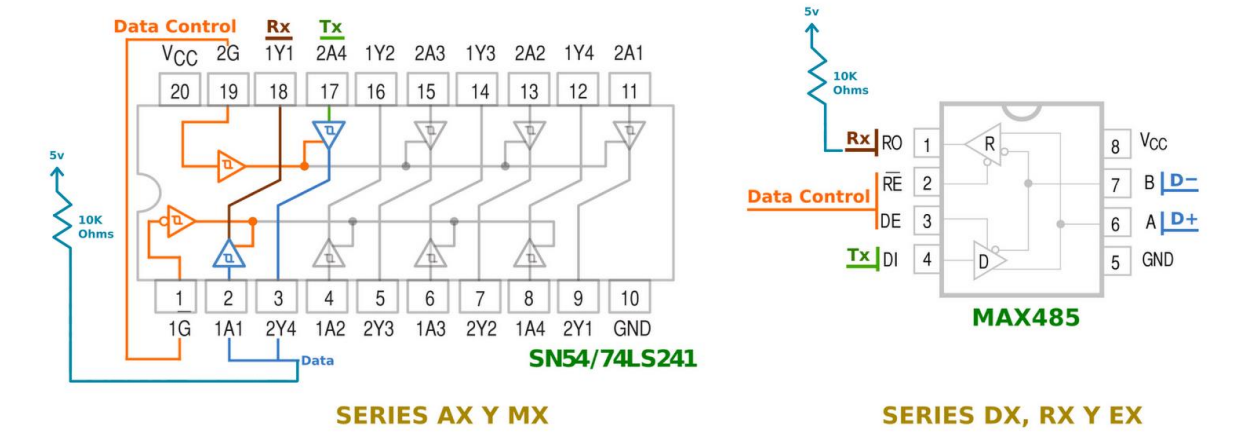


Imagen 3.30 Integrados que se requieren dependiendo el protocolo de comunicación.

En éste proyecto se utilizaron los pines correspondientes al Serial 1, por lo cual, la librería a utilizar es “dynamixelserial1.h” y las señales Data Control, RX y TX que se muestran en la Imagen 3.31 provienen de los pines 2 (Data Control), 18 (TX) y 19 (RX) del Arduino Mega.

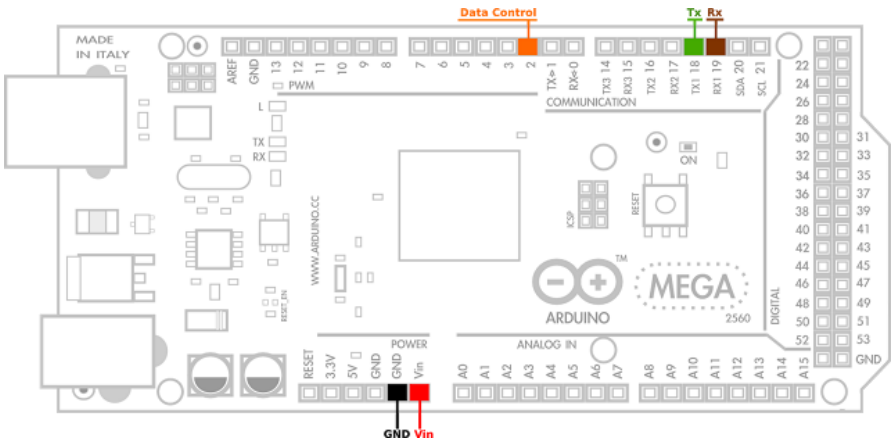


Imagen 3.31 Señales Data Control, RX, y TX.

3.4.2.1 Módulo de comunicación RS-485

En el mercado existe un módulo de comunicación TTL a RS-485 de bajo costo (imagen 3.32), el cual tiene las siguientes características (Mecatronics, 2015):

- Voltaje de Operación: 5 V.
- Consumo Corriente: 500 uA (máx).
- Chip principal: MAX485.
- Tipo de Comunicación: Half-Duplex.
- Velocidad máxima de 10 Mbit/s (a 12 metros).
- Longitud máxima de alcance de 1200 metros (a 100 kbit/s).
- Tamaño de módulo: 35x15x7 mm.

Imagen 3.32 Módulo de comunicación RS-485.

En este proyecto, se utilizó el módulo de comunicación RS485 anteriormente descrito, quedando la conexión Arduino-Dynamixel como lo muestra la imagen 3.33.

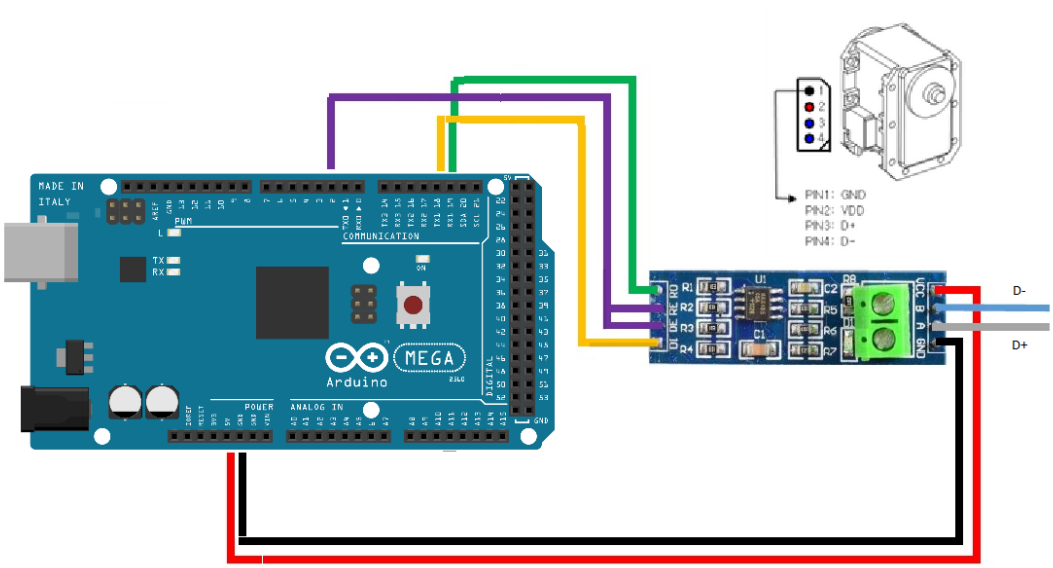


Imagen 3.33 Conexión Arduino-Dynamixel.

Los pines 1 y 2 de los Dynamixel son los pines de alimentación de los servos, dependiendo de la serie y modelo que se vayan a utilizar, el voltaje de operación cambia. Se debe consultar la hoja de especificaciones que proporciona la empresa ROBOTIS para conocer el voltaje de operación recomendado.

NOTA: Es importante juntar la tierra (GND) de la fuente con la cual se alimentarán los servomotores con GND del Arduino.

3.4.2.2 Conexión en serie de los Dynamixel

Una característica de los servomotores Dynamixel, es que el primer servomotor a utilizar debe seguir la conexión descrita en la Imagen 3.33 y en caso de utilizar más de un servomotor sólo se necesita conectar en serie el conector del primer actuador con el segundo y así sucesivamente, cómo se puede apreciar en la imagen 3.34.

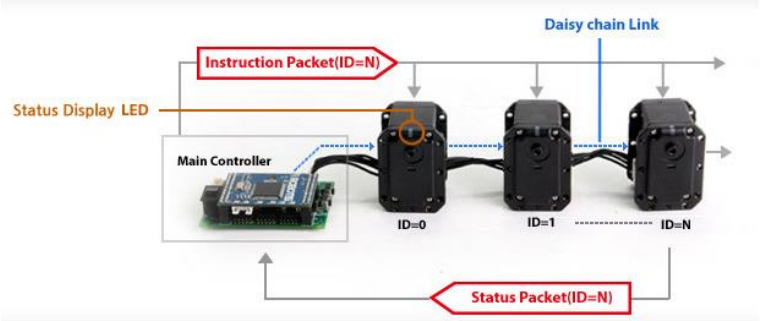


Imagen 3.34 Conexión en serie Dynamixel.

3.4.3 Conexión Arduino – MD25

Para establecer comunicación entre Arduino y la Tarjeta MD25 (imagen 3.35) se pueden usar por dos protocolos de comunicación; ya sea por comunicación Serial o comunicación I^2C. Esto se logra gracias a que la tarjeta cuenta de unos pines para seleccionar el tipo de configuración.

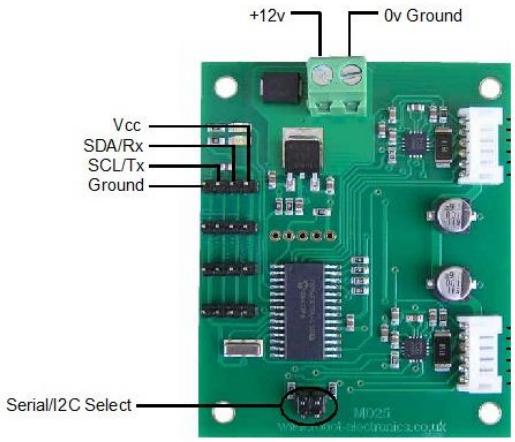


Imagen 3.35 Conexiones MD25.

Es necesario quitar y poner los jumpers en los pines para seleccionar el tipo de comunicación que se desea utilizar, como se puede apreciar en la Tabla 3.8 (R. Electronics, 2011b).

Tabla 3.8 Jumper selección MD25.

Jumper Selection	Protocolo de comunicación	Velocidad de comunicación
	I^2C	Hasta 100 Khz
	Serial	9600 bps

	Serial	19200 bps
	Serial	38400 bps

Cada protocolo de comunicación tiene su documentación y diagrama de conexión. En éste proyecto se utilizó la comunicación I^2C, por lo cual, el diagrama de conexión Arduino – Tarjeta MD25 es como se aprecia en la Imagen 3.36 (R. Electronics, 2011c).

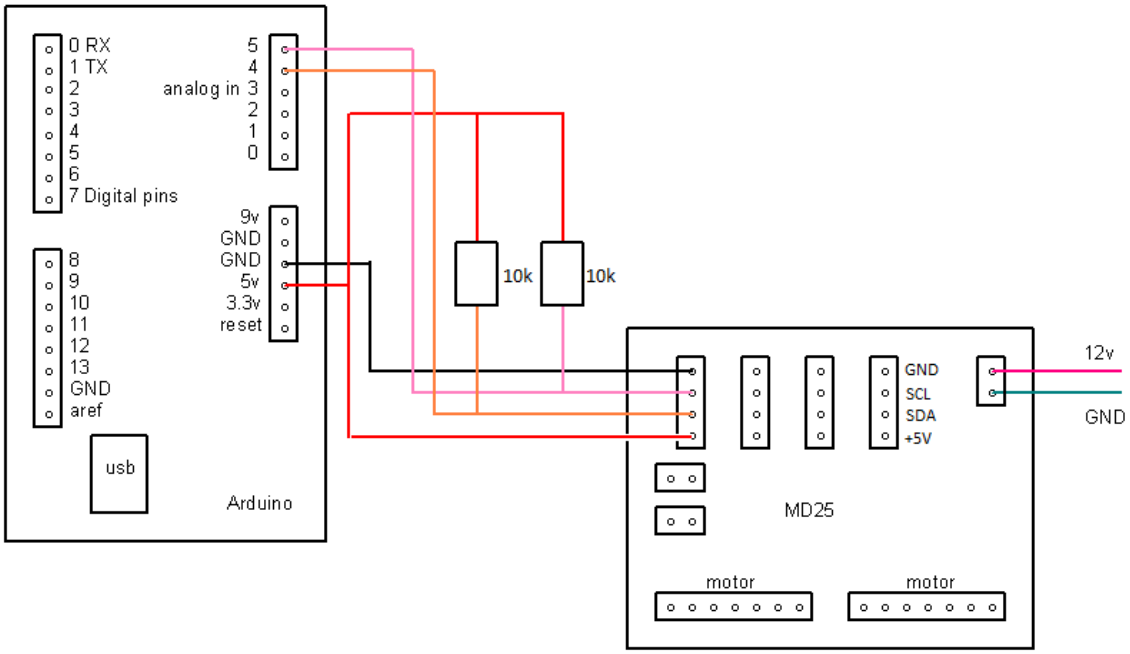


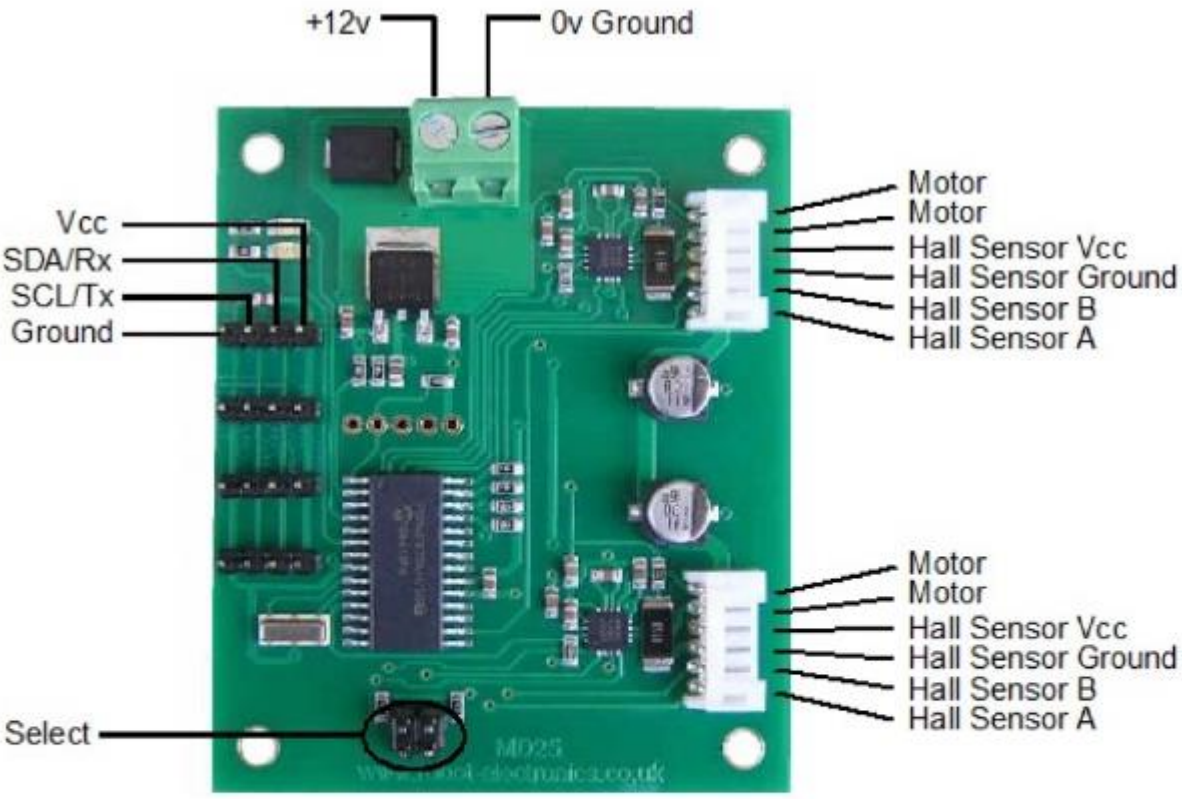
Imagen 3.36Conexión Arduino _MD25 (Modo I^2C).

En la tarjeta Arduino UNO, los pines que se usan como comunicación I^2C son los pines analógicos A4 (SDA) y A5 (SCL), si se quisiera usar otra versión de Arduino, los pines cambiarían como lo establece la Tabla 3.9

Tabla 3.9 Pines de Comunicación I^2C tarjetas Arduino.

Tarjeta	PIN SDA	PIN SCL
Uno, Ethernet	A4	A5
Mega 2560	20	21
Leonardo	2	3
Due	20, SDA1	21, SCL1

Como se ha explicado, la tarjeta MD25 está diseñada para el uso con motores EMG30, sin embargo, en el robot adaptable por cuestiones de tamaño del motor no se usarán esos motores sino unos micro-motorreductores pololu, sin embargo, la información que nos proporciona la MD25 es de gran utilidad para el robot adaptable, por lo cual es necesario el motor Pololu con encoder para poder hacer los ajustes pertinentes y lograr que la tarjeta MD25 se pueda comunicar con los encoders de los micro-motorreductores, como lo muestra la Imagen 3.37.



Wire colour	Connection
Purple (1)	Hall Sensor B Vout
Blue (2)	Hall sensor A Vout
Green (3)	Hall sensor ground
Brown (4)	Hall sensor Vcc
Red (5)	+ Motor
Black (6)	- Motor

Imagen 3.37 Conexión MD25 – micro-motorreductores POLOLU.

Capítulo 4

Diseño de software para el banco de pruebas

Se entiende como software a “cualquier programa de computadora, la documentación asociada y configuración de datos que se necesitan para hacer que esos programas operen de manera correcta” (Sommerville & Galipienso, 2005). El software debe ser funcional y cumplir los aspectos requeridos por el usuario, además de ser confiable, fácil de usar y de fácil mantenimiento.

Existen distintos modelos de procesos para el desarrollo de software, tendiendo entre los dos más conocidos e implementados al de *cascada* y el *iterativo.* El *modelo de cascada* considera las actividades de especificación, desarrollo y evolución como etapas de procesos separados (Sommerville & Galipienso, 2005).

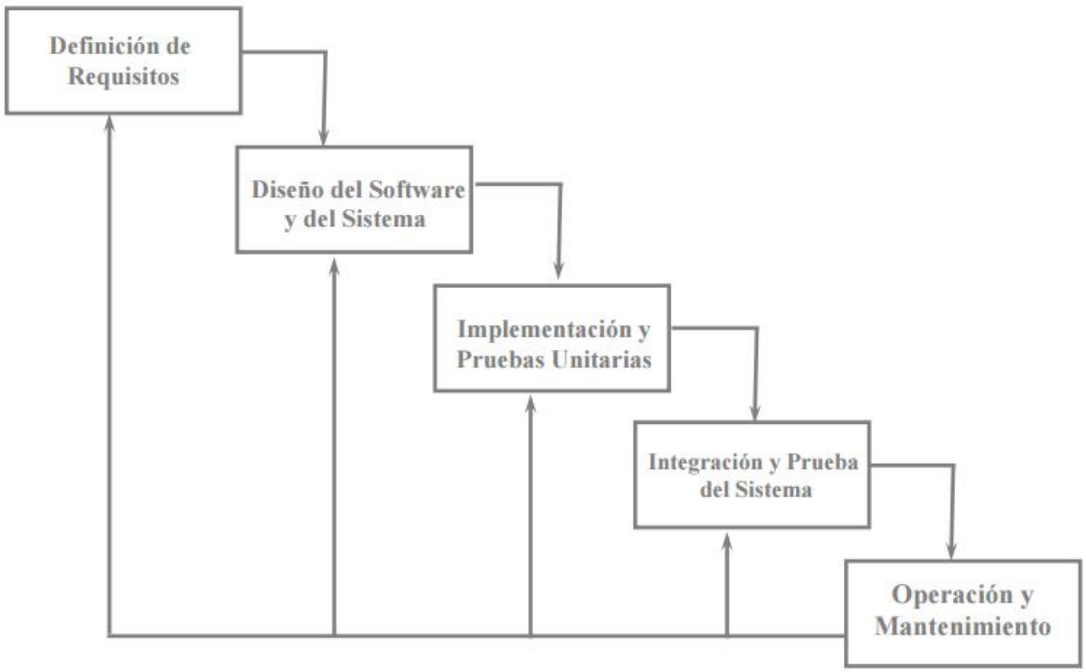


Imagen 4.1 Diagrama modelo cascada.

El *modelo iterativo* consiste en entrelazar las actividades de especificación, desarrollo y evolución, es decir, se desarrolla rápidamente un sistema inicial a partir de especificaciones muy abstractas y se va refinando el sistema basándose en las peticiones del cliente hasta que se satisfacen las necesidades del cliente.

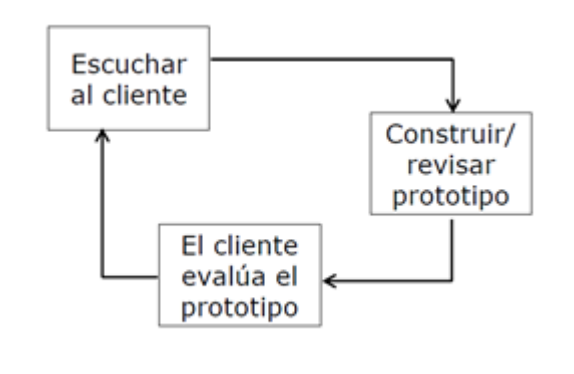


Imagen 4.2 Diagrama modelo iterativo.

Considerando al costo total del desarrollo del software en una escala de 0 a 100, la siguiente grafica muestra cómo se gastaría el presupuesto en las diferentes actividades del proceso según el modelo que se implemente:

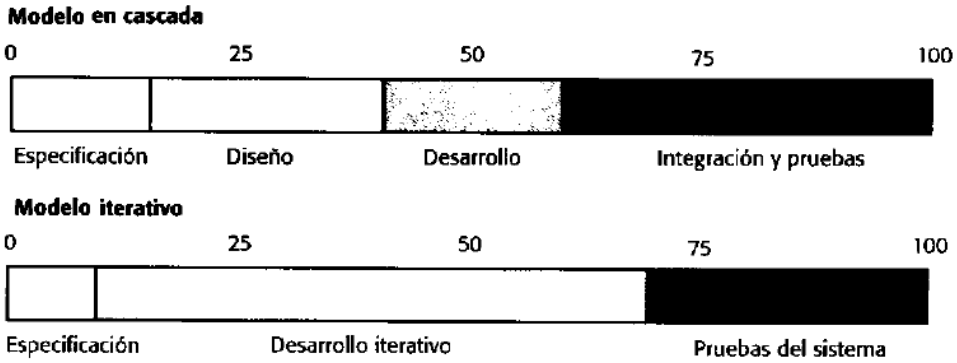


Imagen 4.3 Comparación de recursos (Modelo cascada y modelo iterativo).

Utilizando el modelo en cascada los costos de especificación, diseño, desarrollo e integración y pruebas del sistema se miden de forma separada, observando que las etapas de integración y pruebas son las actividades del desarrollo que más recursos consumen. A diferencia el modelo iterativo no divide ninguna de las etapas entre el diseño y desarrollo del software, ya que las actividades que conllevan estas fases se llevan a cabo en paralelo, dejando al final sólo la etapa de prueba del sistema final.

Tabla 4.1. Comparación de modelo cascada y modelo iterativo.

Modelo en cascada	Modelo iterativo
• Modelo de proceso clásico (desde los 70´s) • Basado en la mentalidad de línea de ensamblaje • Es sencillo y fácil de entender • El proyecto pasa a través de una serie de fases (para poder pasar a la otra fase se tiene que haber conseguido todos los objetivos de la fase anterior) Ventajas: • Es un método sencillo Desventajas: • No se ve un producto hasta muy avanzado el proceso. • Se necesita que las especificaciones sean estables, es decir, que no cambien. • Las revisiones son de gran complejidad, ya que si una revisión no se realiza de manera adecuada se necesita regresar al proceso anterior.	• Comienza con la recolección de requerimientos. (Cliente y desarrolladores definen las especificaciones globales del sistema.) • Diseño rápido centrado en los aspectos visibles para el cliente. (Se construye un prototipo.) • El prototipo lo evalúa el cliente y lo utiliza para refinar los requisitos. • El proceso se repite Ventajas: • Se tiene rápidamente un software que muestra el funcionamiento general del producto • Las especificaciones pueden ser dinámicas, es decir, pueden ir cambiando. Desventajas: • Si el cliente pide unos ajustes probablemente se tenga que desechar completamente el prototipo o solo hacerle unas mejoras.
NOTA: Este método es aplicable cuando se tienen muy bien definidas las especificaciones.	NOTA: Este método se utiliza cada vez más cuando la rapidez del desarrollo es esencial.

También se debe considerar en el proceso de realización de un software un conjunto de actividades y resultados asociados, el cual se puede dividir en 3 fases:

Tabla4.2. Descripción del proceso de software.

Proceso de software	Fase 1 (Especificación)	Especificación	Establecer requerimientos y especificaciones
	Fase 2 (Desarrollo)	Diseño	Producir un boceto del sistema
		Implementación	Construcción del software
		Validación	Hacer pruebas para ver que el sistema cumple con las especificaciones
		Instalación	Entregar el sistema al usuario
	Fase 3 (Evolución)	Mantenimiento	Reparar fallos en el sistema Adaptar o mejorar el sistema

En cuanto a la fase 3 de evolución, y en concreto con la especificación del mantenimiento podemos identificar cuatro tipos del mismo:

1. *El mantenimiento de corrección*. El cual consiste en corregir los defectos encontrados en el sistema.
2. *El mantenimiento de adaptación*. Este surge de las modificaciones que se necesiten realizar por un cambio externo al sistema.
3. *El mantenimiento de mejora.* Se basa en ampliar los requisitos funcionales originales, a petición del cliente.
4. *El mantenimiento de prevención***.** Consiste en una atención constante de limpieza, revisión y afinación de los distintos elementos integrantes de un equipo de cómputo. Es necesario darle mantenimiento al software ya que el continuo uso genera una serie de cambios en la configuración original del sistema, causando bajas en el rendimiento que al acumularse con el tiempo pueden generar problemas serios. Actualmente es indispensable mantener actualizada la protección contra virus informáticos.

4.1 Fase 1: Especificación

En la Fase 1 en el diseño del software, se establecen los requerimientos y posteriormente se determinan las especificaciones. El *requerimiento* es el deseo que expresa en términos vagos y comunes el cliente potencial, se presenta en lenguaje coloquial. Para la *especificación* hay que llevar a cabo un procesamiento de la información, de tal manera que se pueda expresar los requerimientos en términos de una manera clara, lenguaje ingenieril.

En éste proyecto los requerimientos son:

1. Crear una herramienta en SIMULINK que permita controlar a los elementos que conforman el robot adaptable, pero dicha herramienta deberá poder ser utilizada en cualquier proyecto del Mechatronic Research Group.
2. Crear un módulo en SIMULINK para los servomotores Dynamixel y otro módulo para la tarjeta MD25 que sean capaz de enviar y recibir datos.
3. Cada módulo deberá ser capaz de controlar todos los elementos que se configuren y se utilicen en los proyectos, es decir, el bloque de Dynamixel deberá poder establecer comunicación con la cantidad de servos que se utilicen y de manera similar el bloque para la MD25.
4. Los datos que se obtengan de cada uno de los elementos servirán para conformar un banco de pruebas para un robot adaptable,

Como se pudo apreciar en el subtema 3.4.2.2 *(Conexión en serie de los Dynamixel)* y el subtema 2.7 *(Tarjeta MD25)* una de las características de éstos elementos es que se pueden establecer comunicación con distintos elementos sólo modificando un parámetro. En el caso de los servomotores Dynamixel al cambiar el ID se pueden controlar hasta 254 servomotores. En el caso de la Tarjeta MD25 al cambiar la dirección de comunicación se pueden controlar hasta 8 Tarjetas MD25.

Con base a los requerimientos anteriormente descritos y consultando la documentación que proporcionan los fabricantes Robotis (Dynamixel) y Robot Electronics (MD25) se establecieron las siguientes especificaciones:

Tabla 4.3 Especificaciones de los modulos se SIMULINK.

Especificaciones Dynamixel	Especificaciones MD25
Diseñar modulo en SIMULINK con las siguientes características: 1. Número de entradas: 3 a. ID b. Posición deseada c. Velocidad deseada 2. Número de salidas: 6 a. ID b. Posición c. Velocidad d. Carga i. Sentido de giro ii. Porcentaje de carga	Diseñar modulo en SIMULINK con las siguientes características: 1. Número de entradas: 3 a. Dirección de la tarjeta b. Velocidad del motor A c. Velocidad del motor B 2. Número de salidas: 6 a. Dirección de la tarjeta b. Encoder acoplado al motor A c. Encoder acoplado al motor B d. Voltaje e. Corriente motor A f. Corriente motor B

e. Voltaje f. Temperatura 3. Tipo de comunicación: Serial - RS-485	3. Tipo de comunicación: Serial – I^2C

El método que se eligió para el desarrollo de estos módulos en SIMULINK, fue el modelo iterativo, ya que permite la obtención rápida de un producto utilizable, además, tener ajustes inmediatos que se pueden realizar tomando en cuenta las necesidades del usuario.

4.2 Fase 2: Desarrollo

De acuerdo a las especificaciones, se requiere el diseño y creación de dos módulos en SIMULINK que permita controlar tanto a los servomotores Dynamixel y la Tarjeta MD25.

SIMULINK es un entorno de programación visual que funciona sobre el entorno de programación de MATLAB. MATLAB es un ambiente de programación para el desarrollo de algoritmos, análisis de datos, visualización y cómputo numérico. (MATLAB, 2013) Usando MATLAB, se pueden resolver problemas de cómputo técnico más rápido que los lenguajes de programación como C/C++, Fortran, entre otros.

SIMULINK es un ambiente para simulación multidominio y diseño basado en modelos para sistemas dinámicos y embebidos. Proporciona un ambiente gráfico y un conjunto de librerías de bloques personalizables que le permiten, diseñar, simular y probar una variedad de sistemas cambiantes en el tiempo, incluyendo comunicaciones, control, procesamiento de señales, video e imágenes.

En este proyecto, se utilizó SIMULINK como una herramienta de comunicación con Arduino mediante comunicación Serial. La tarjeta Arduino establece comunicación con los servomotores mediante comunicación RS-485 y con los micro-motorreductores Pololu con comunicación I^2C; finalmente Arduino envía a SIMULINK las lecturas obtenidas de los elementos anteriormente descritos por comunicación Serial. A continuación, se explican las características principales de los tipos de comunicación que se utilizaron.

4.2.1 Comunicación Serial

Hoy en día la manera más común de comunicación entre dispositivos electrónicos es la *Comunicación Serial* y Arduino no es la excepción. A través de este tipo de comunicación se pueden enviar y recibir datos desde nuestro Arduino a otros microcontroladores o una computadora.

Un puerto serie envía la información mediante una secuencia de bits. Para ello se necesitan al menos dos conectores para realizar la comunicación de datos RX (Recepción) y TX (Transmisión), imagen 4.4 (Llamas, 2014).

Imagen 4.4 Comunicación serial.

En ocasiones se hace referencia a los puertos serie como UART (Universally Asynchronous Receiver Transmitter), es una unidad que incorporan ciertos procesadores, encargada de realizar la conversión de los datos a una secuencia de bits y transmitirlos o recibirlos a una velocidad determinada.

También se utiliza el término TTL (Transistor-Transistor Logic), esto significa que la comunicación se realiza mediante variaciones en la señal entre 0 V y V_{CC}, donde V_{CC} suele ser de 3.3 V o 5 V.

Prácticamente todas las placas de Arduino disponen de al menos una unidad UART. Las placas Arduino UNO y Mini Pro disponen de una unidad UART que opera a nivel TTL 0 V – 5 V. La placa Arduino MEGA cuenta con 4 unidades UART.

El modo más sencillo y común de comunicación serial es asíncrona (8 bits, 1 bit de parada), siempre se está enviando un byte, es decir, un tren de 8 pulsos de voltaje legible por los dispositivos como una serie de 8 bits de (1 o 0).

En Arduino, existen funciones para escribir por el puerto serial en distintos formatos, conocidos como modificador: decimal (DEC), hexadecimal (HEX), octal (OCT), binario (BIN) o byte (BYTE) (Arduino, 2016c). No importa que modificador se utilice, siempre se están enviando bytes, la diferencia está en lo que esos bytes van a representar y sólo hay dos opciones:

1. Una serie de caracteres ASCII.
2. Un número.

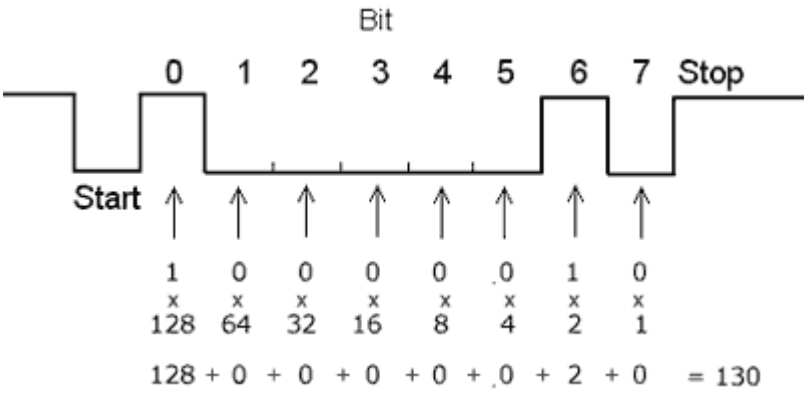


Imagen 4.5 Transmisión de información Comunicación Serial.

En la tabla 4.4 se muestra la forma en que Arduino envía la información dependiendo el modificador que se seleccione.

Tabla 4.4. Transmisión de información Comunicación Serial.

Dato	Modificador	Conversión	ASCII	Pulsos	Envío de pulsos
65	DEC	65	54 - 55	2	000110110 - 000110111
65	HEX	41	52 - 59	2	000110100 - 000110001
65	OCT	101	49 - 48 - 49	3	000110001 - 000110000 - 000110001
65	BIN	01000001	49-48-49-49-49-49-49-48	8	000110000 – 000110001-000110000 - 000110000-000110000 - 000110000-000110000 - 000110001
65	BYTE	01000001	NA	1	010000001

Es evidente como el modificador BYTE permite él envió de información más económica (menos pulsos para la misma cantidad de información), lo que implica mayor velocidad en la comunicación y esto es importante cuando se piensa en interacción en tiempo real.

Hay que tener en cuenta las limitaciones de la transmisión en la comunicación serie, sólo se realiza a través de valores con una longitud de 8 bits (1 byte).

4.2.2 Comunicación RS-485

Cuando se necesita trasmitir información a altas velocidades en largas distancias, se piensa en la comunicación RS-485. En este tipo de comunicación se pueden conectar hasta 32 nodos

con sólo un par de cables, dependiendo de la distancia, circuitos integrados utilizados y de la velocidad de transmisión.

La comunicación RS-485 presenta las siguientes ventajas con respecto a la comunicación RS-232(Pérez):

1. Bajo costo

Se dice que es de bajo costo porque sólo se necesita una fuente de 5 volts para hacer que los circuitos integrados puedan recibir y transmitir información, a diferencia de una comunicación RS-232, donde muchos de sus circuitos integrados necesitan diferentes voltajes de operación y esto provoca que se necesite más de una fuente de alimentación.

2. Capacidad de interconexión

La comunicación RS-485 es multi-enlace, es decir que podemos tener múltiples transmisores y receptores, lo que nos da la posibilidad de poder llegar a tener hasta un máximo de 256 nodos.

3. Longitud de enlace

Con la comunicación RS-232 es posible transmitir datos a lo largo de entre 50 y 100 ft de longitud (aprox. 15 m y 30 m), en contraste con la comunicación RS-485, donde se puede llegar a tener un buen enlace hasta una longitud de 4000 ft (aprox. 1219 m).

4. Rapidez

La transmisión de los bits puede ser de hasta 10 Megabits/segundo.

4.2.3 Comunicación I^2C

La comunicación I^2C (Inter-Integrated Circuits) es un bus de comunicación muy utilizado para la comunicación entre circuitos integrados, uno de sus usos más comunes es la comunicación entre un microcontrolador y sensores periféricos (UAH, 2000).

Características:

- Bus de comunicación síncrono
- Bus formado por dos hilos
    - SDA (Serial Data line): Datos
    - SCL (Serial Clock line): Reloj
    - Vcc y GND

- Velocidad de transmisión:
    - Standard: hasta 100 kbits/s
    - Fast: hasta400 kbits/s
    - High-Speed: hasta 3.4 Mbits/s.
- Cada dispositivo del bus tiene una dirección única
- Distancia y número de dispositivos
    - Limitado por la capacidad del bus. Normalmente 2 o 3 [m]
- Protocolo de acceso al bus
    - Maestro – esclavo
    - Multimaestro

Ventajas:

- Pocos cables de interconexión.
- Conexión de dispositivos a distancia.

Desventajas:

- Disponibilidad de circuitos que soporten el bus.

En la Imagen 4.6 se muestra la conexión de los dispositivos al bus (Jesús, 2014).

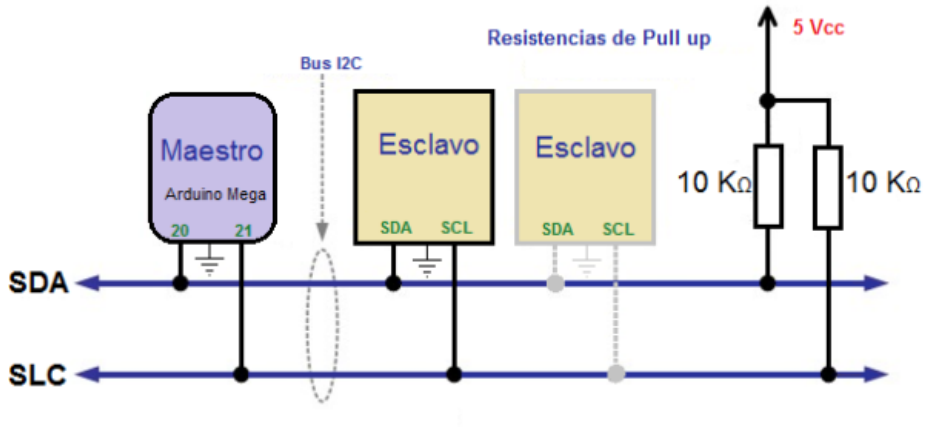


Imagen 4.6 Comunicación I^2C.

El bus es activo bajo, por tal motivo se conectan resistencias en modo Pull-up, es decir, la señal activa es un 0. Si le llega un nivel alto (1) estará abierto el bus de comunicación; si le llega un nivel bajo (0) estará cerrada el bus de comunicación (imagen 4.7).

Imagen 4.7 Modo activo bajo.

La misma línea de datos envía la información en las dos direcciones (half-duplex), por lo que es necesario un control de acceso al bus y un direccionamiento de cada elemento.

Algunas de las características del dispositivo maestro son las siguientes:

- Controla la comunicación debido a que es el que genera la señal de reloj (SCL).
- Inicia y termina la comunicación.
- Direcciona a los esclavos.
- Establece el sentido de la comunicación.

Estructura de la comunicación I^2C

- *Bit de Start*: Este bit provoca un cambio de 1 a 0 cuando SCL está a nivel alto. *(Master)*
- *Dirección:* El primer byte enviado empieza con 7 bits de dirección, el cual indica a quien enviamos o solicitamos el dato. *(Master)*
- *R/W (Leer/Escribir):* El siguiente bit indica si vamos a realizar una operación de lectura o escritura. *(Master)*
- *ACK:* Este bit está presente al final de cada byte que enviamos y nos permite asegurarnos que el byte ha llegado a su destino. De este modo el que envía deja el bit a 1 y si alguien ha recibido el mensaje fuerza ese bit a 0. De esta manera confirma que le ha llegado el byte y la transmisión puede continuar *(Slave).*
- *Byte de Datos:* Aquí ponemos el dato que queramos escribir o leer *(Master/ Slave).*
- Se espera un *ACK* del receptor *(Master/ Slave).*
- Se repiten los dos últimos pasos tantas veces como sea necesario.
- *Bit de Stop:* Este bit provoca el paso de 0 a 1 cuando la línea *SCL* se encuentra en alto. Esto termina la transmisión y deja el bus libre para que otro puede empezar a transmitir *(Master).*

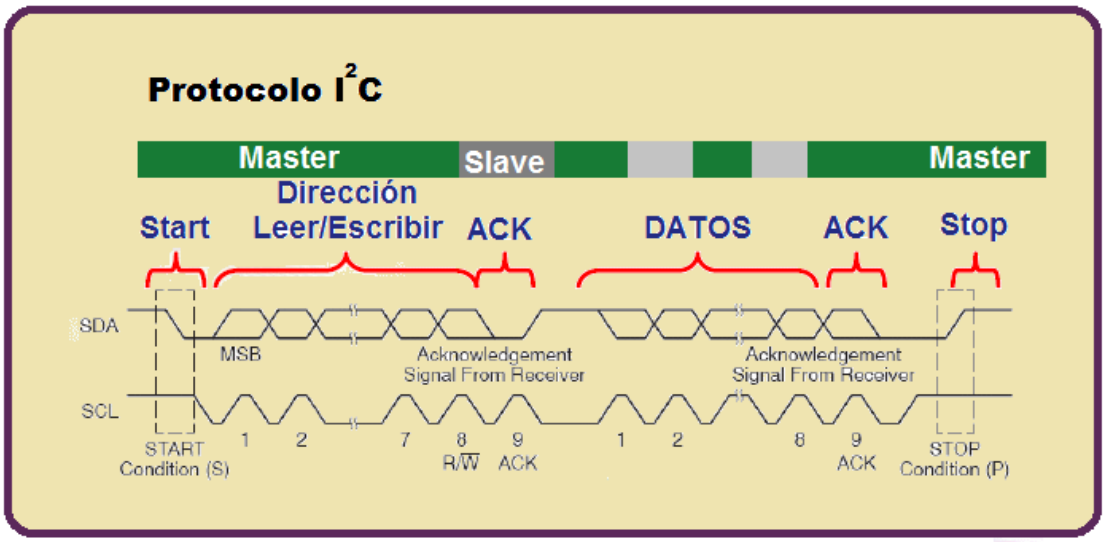


Imagen 4.8 Estructura de la comunicación I²c.

Arduino cuenta con una librería llamada "Wire.h" (Arduino, 2016d), la cual es la encargada de gestionar este protocolo de comunicación, los pines que se utilizan como SCL y SDA dependen de la tarjeta a utilizar como se mencionó en la Tabla 3.9.

Ya que se conoce cómo funciona de manera general los distintos tipos de comunicación y considerando las especificaciones que se establecieron en la Tabla 4.3. Se propone utilizar los siguientes bloques para establecer la comunicación entre SIMULINK y Arduino (imagen 4.9).

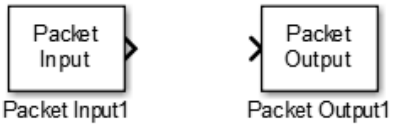


Imagen 4.9 Bloques de SIMULINK para enviar y recibir datos.

Sin embargo, una configuración que es importante realizar para poder utilizar los bloques de *packet input* y *packet output* es configurar MATLAB en tiempo real, lo cual se logra, introduciendo el siguiente comando ***sldrtkernel –install (***versión 2015 de MATLAB) en el Command Window de MATLAB como se muestra en la imagen 4.10 (Si es una versión inferior se debe introducir *rtwintgt –install*).

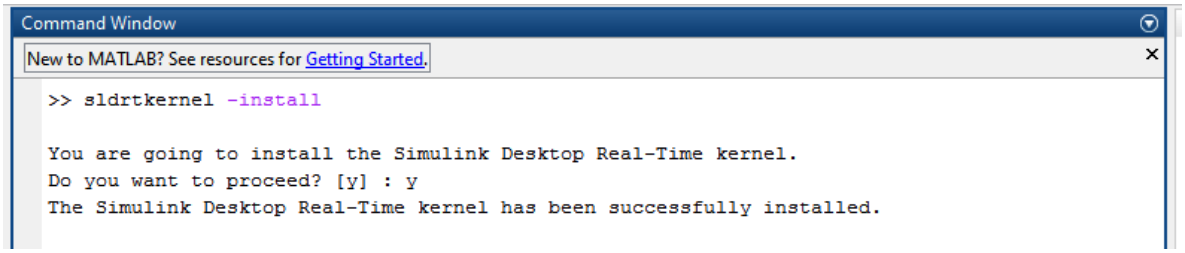


Imagen 4.10 Configurar MATLAB en tiempo real.

Los bloques de paquetes de entrada y salida, tienen la ventaja de poder configurarse con distintos protocolos de comunicación, lo cual permite que sin cambiar el bloque en SIMULINK podamos enviar y recibir información de un puerto serial, puerto paralelo, protocolo UDP (Wi-Fi), etc. En este proyecto se usa como puerto serial para comunicarse con Arduino.

Para la configuración del *packet input* o *packet output*, se debe hacer doble clic izquierdo sobre alguno de los bloques y aparecerá un cuadro de configuración, lo primero que se hace es dar clic sobre *Install new board -> Standard devices -> Serial port* como se muestra en la Imagen 4.11.

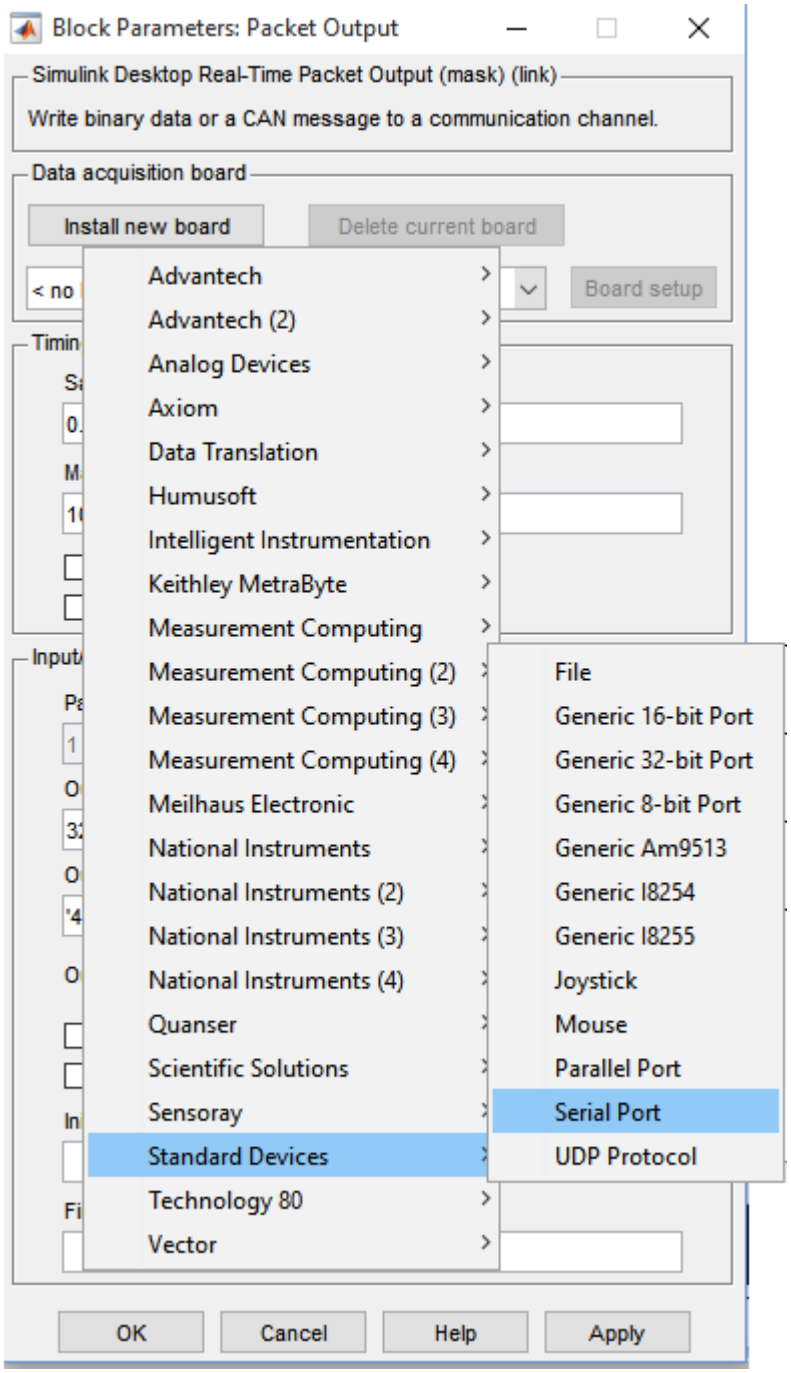


Imagen 4.11 Procedimiento para seleccionar el protocolo de comunicación.

Al seleccionar *Serial Port*, aparecerá un nuevo cuadro de configuración en donde se deberán establecer los parámetros con los cuales se realizará la comunicación serial. Dos parámetros importantes de este cuadro de configuración es el puerto con el cual se establecerá la comunicación y la velocidad de transmisión (*Baudrate*).

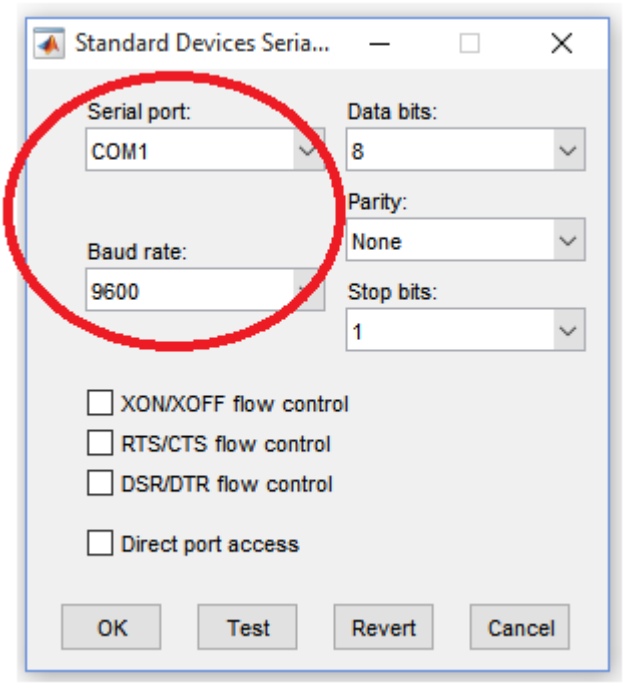


Imagen 4.12 Configuración de la comunicación Serial.

Para la comunicación con Arduino los demás parámetros no se modifican, de esta manera queda configurado a que tarjeta le mandaremos la información o de cual recibiremos la información.

Como se había visto en la Tabla 4.4, la manera más eficiente de mandar información por el puerto serial es por medio de bytes. Por lo cual es importante definir cuantos bytes se enviarán en el caso de *Packet Output* y cuantos bytes se recibirán en el caso de *Packet Input*, lo anterior se logra modificando los siguientes parámetros "Packet size" y "Data types", como se muestra en la Imagen 4.13.

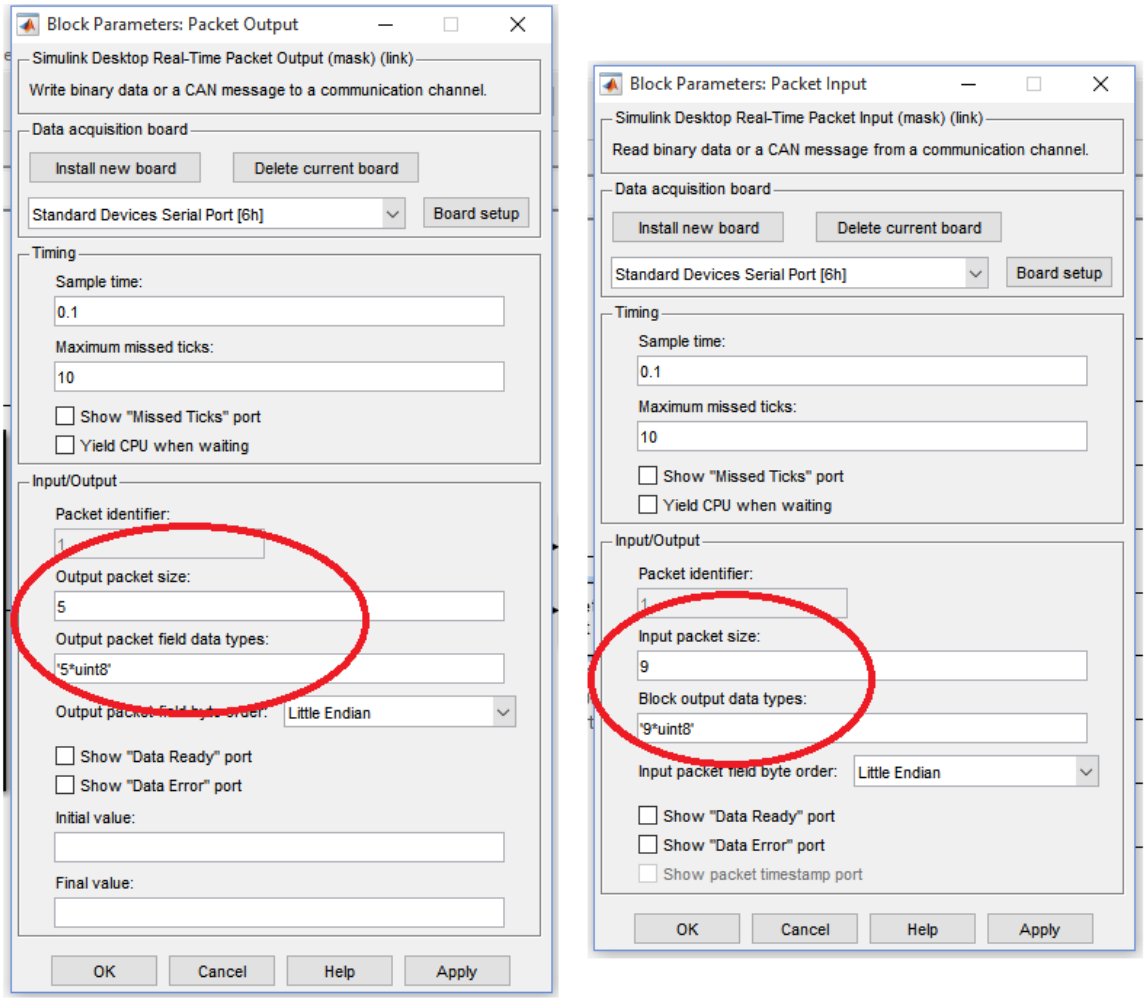


Imagen 4.13 Configuración de la cantidad de bytes para enviar y recibir.

El parámetro “packet size” indica la cantidad de bytes a enviar o recibir; el parámetro “Data types” indica el número de datos y el formato que tendrá el dato. El ejemplo de la Imagen 4.13 indica que se enviaran 5 bytes los cuales son 5 datos en formato unit8; y se recibirán 9 bytes los cuales son 9 datos en formato unit8.

Los tipos de datos que se pueden manipular en MATLAB son los que se muestran en la Tabla 4.5 (MathWorks, 2016):

Tabla 4.5 Tipos de datos numéricos que se pueden utilizar en MATLAB.

Numeric Data Types	Descripción	Numero de bits	Numero de bytes
double	Flotante de doble precisión	64 bits	8 bytes
single	Flotante de simple precisión	32 bits	4 bytes
int8	Entero con signo	8 bits	1 byte
int16	Entero con signo	16 bits	2 byte
int32	Entero con signo	32 bits	4 bytes
int64	Entero con signo	64 bits	8 bytes
uint8	Entero sin signo	8 bits	1 byte
uint16	Entero sin signo	16 bits	2 bytes
uint32	Entero sin signo	32 bits	4 bytes
uint64	Entero sin signo	64 bits	8 bytes

De inicio, los bloques vienen configurados como packet size = 32 y data types = '4*double', Para calcular el tamaño del paquete se requiere multiplicar la cantidad de datos a enviar por los bytes que requiere el tipo de dato, es decir, para enviar 2 datos en formato uint16, el tamaño del paquete se calcula como lo muestra la tabla 4.6.

Tabla 4.6

Datos	Tipo de dato	
2	uint16	Packet SIze
2	2 bytes	4

Y se expresa de la siguiente manera

Packet size = 4 y data types = '2*uint16'

Para determinar estos parámetros para el diseño de los módulos a desarrollar es necesario consultar la documentación de los Dynamixel y la Tarjeta MD25 las cuales se pueden consultar de manera detallada en los Apéndices 5 (R. Electronics, 2011a) y 6 (ROBOTIS, 2010a).

4.2.4 Diseño del módulo SIMULINK (Dynamixel)

Para los servomotores se tiene la tabla de control, la cual contiene datos relativos a la situación actual y el funcionamiento del servomotor, el usuario puede controlar el Dynamixel interactuando con dichos datos. La tabla está dividida en 2 partes (EEPROM y RAM). Básicamente los datos de la memoria EEPROM son datos de configuración del servomotor y los datos de la memoria RAM que son los que permiten controlar y leer las variables de interés.

En las especificaciones se determinaron como parámetros importantes para enviar a los Dynamixel: 1) ID del servomotor, 2) Posición deseada, 3) Velocidad deseada. Estos parámetros se encuentran en la tabla de control como se muestra en la Tabla 4.7.

Tabla 4.7 Tabla de control Dynamixel. Selección de los parámetros para enviar.

Area	Address (Hexadecimal)	Name	Description	Access	Valor inicial (Hexadecimal)
	24 (0X18)	Torque Enable	Torque On/Off	RW	0 (0X00)
	25 (0X19)	LED	LED On/Off	RW	0 (0X00)
	26 (0X1A)	CW Compliance Margin	Margen de cumplimiento sentido horario	RW	1 (0X01)
	27 (0X1B)	CCW Compliance Margin	Margen de cumplimiento sentido antihorario	RW	1 (0X01)
	28 (0X1C)	CW Compliance Slope	Cumplimiento de inclinación sentido horario	RW	32 (0X20)
	29 (0X1D)	CCW Compliance Slope	Cumplimiento de inclinación sentido antihorario	RW	32 (0X20)
	30 (0X1E)	Goal Position(L)	Posicion meta	RW	-
	31 (0X1F)	Goal Position(H)	Posicion meta	RW	-
	32 (0X20)	Moving Speed(L)	Velocidad de movimiento	RW	-
	33 (0X21)	Moving Speed(H)	Velocidad de movimiento	RW	-
	34 (0X22)	Torque Limit(L)	Limite de par (Goal Torque)	RW	ADD14
	35 (0X23)	Torque Limit(H)	Limite de par (Goal Torque)	RW	ADD15
RAM	36 (0X24)	Present Position(L)	Posición actual	R	-
	37 (0X25)	Present Position(H)	Posición actual	R	-
	38 (0X26)	Present Speed(L)	Velocidad actual	R	-
	39 (0X27)	Present Speed(H)	Velocidad actual	R	-
	40 (0X28)	Present Load(L)	Carga actual	R	-
	41 (0X29)	Present Load(H)	Carga actual	R	-
	42 (0X2A)	Present Voltage	Voltaje actual	R	-
	43 (0X2B)	Present Temperature	Temperatura actual	R	-
	44 (0X2C)	Registered	La instrucción se ha registrado	R	0 (0X00)
	46 (0X2E)	Moving	¿Hay akgun movimiento?	R	0 (0X00)
	47 (0X2F)	Lock	Blockeo EEPROM	RW	0 (0X00)
	48 (0X30)	Punch(L)	Punch	RW	32 (0X20)
	49 (0X31)	Punch(H)	Punch	RW	0 (0X00)

1. ID: Indica a que servo queremos manipular. Se pueden configurar de 0 a 253.
2. Posición deseada: Se puede mover de 0° a 300°. Como se muestra en la Imagen 4.14.
3. Velocidad deseada: Se puede mover el servo en valores de 0 a 1023, en donde cada valor equivale a 0.111 RPM, por ejemplo, si se enviar un 300 se trata de 33.3 RPM, si se envía un 1023 se trata de 114 RPM.

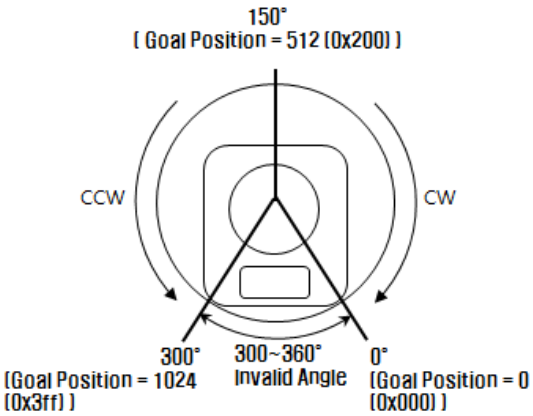


Imagen 4.14 Funcionamiento de la posición Dynamixel serie RX.

Para el diseño del módulo en SIMULINK se optó por enviar los datos de la siguiente manera (tabla 4.8):

Tabla 4.8 Cantidad de bytes a enviar desde SIMULINK módulo Dynamixel.

Parámetro	Rango de operación	Numero de bytes
ID	0 - 253	1 byte
Posición deseada	0 - 300	2 bytes
Velocidad deseada	0 - 1023	2 bytes

Como se puede apreciar en la Tabla 4.7 se requieren enviar 5 bytes de información en formato uint8, es decir, los parámetros para el packet output son los siguientes: packet size = 5 y data type = ‘5*uint8’ como se muestra en la Imagen 4.15:

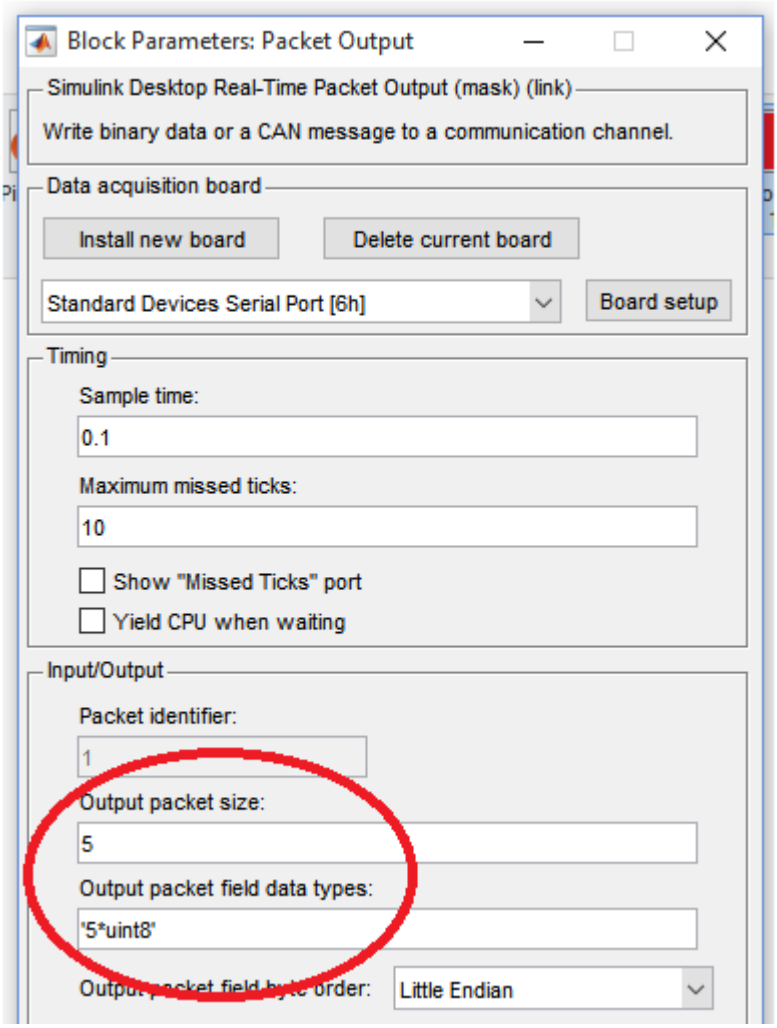


Imagen 4.15 Configuración de los bytes de envió SIMULINK-Dynamixel.

En las especificaciones se determinaron como parámetros importantes para leer de los Dynamixel: 1) ID del servomotor, 2) Posición actual, 3) Velocidad actual, 4) Carga, 5) Voltaje, 6) Temperatura. Estos parámetros se encuentran en la tabla de control y se debe consultar de qué manera funcionan, como se muestra en la Tabla 4.9.

Tabla 4.9 Tabla de control Dynamixel. Selección de los parámetros para recibir.

Area	Address (Hexadecimal)	Name	Description	Access	Valor inicial (Hexadecimal)
	24 (0X18)	Torque Enable	Torque On/Off	RW	0 (0X00)
	25 (0X19)	LED	LED On/Off	RW	0 (0X00)
	26 (0X1A)	CW Compliance Margin	Margen de cumplimiento sentido horario	RW	1 (0X01)
	27 (0X1B)	CCW Compliance Margin	Margen de cumplimiento sentido antihorario	RW	1 (0X01)
	28 (0X1C)	CW Compliance Slope	Cumplimiento de inclinación sentido horario	RW	32 (0X20)
	29 (0X1D)	CCW Compliance Slope	Cumplimiento de inclinación sentido antihorario	RW	32 (0X20)
	30 (0X1E)	Goal Position(L)	Posicion meta	RW	-
	31 (0X1F)	Goal Position(H)	Posicion meta	RW	-
	32 (0X20)	Moving Speed(L)	Velocidad de movimiento	RW	-
	33 (0X21)	Moving Speed(H)	Velocidad de movimiento	RW	-
	34 (0X22)	Torque Limit(L)	Limite de par (Goal Torque)	RW	ADD14
	35 (0X23)	Torque Limit(H)	Limite de par (Goal Torque)	RW	ADD15
RAM	36 (0X24)	Present Position(L)	Posición actual	R	-
	37 (0X25)	Present Position(H)	Posición actual	R	-
	38 (0X26)	Present Speed(L)	Velocidad actual	R	-
	39 (0X27)	Present Speed(H)	Velocidad actual	R	-
	40 (0X28)	Present Load(L)	Carga actual	R	-
	41 (0X29)	Present Load(H)	Carga actual	R	-
	42 (0X2A)	Present Voltage	Voltaje actual	R	-
	43 (0X2B)	Present Temperature	Temperatura actual	R	-
	44 (0X2C)	Registered	La instrucción se ha registrado	R	0 (0X00)
	46 (0X2E)	Moving	¿Hay akgun movimiento?	R	0 (0X00)
	47 (0X2F)	Lock	Blockeo EEPROM	RW	0 (0X00)
	48 (0X30)	Punch(L)	Punch	RW	32 (0X20)
	49 (0X31)	Punch(H)	Punch	RW	0 (0X00)

1. ID: Muestra de que servo se está leyendo la información. (0 – 253).
2. Posición actual: Funciona de manera análoga a la posición deseada. Indica en qué posición se encuentra el servo cuando se hace la lectura (0° - 300°).
3. Velocidad actual: Funciona de manera análoga a velocidad deseada. Indica a qué velocidad se está moviendo el servo cuando se hace la lectura (0 - 1023)
4. Carga actual: Este valor tiene un rango de operación (0 – 2047), dicho valor tiene dos significados: el porcentaje de carga (0% - 100%) y el sentido en que la carga está trabajando (Sentido horario CW o sentido anti horario CCW)
    a. Valores de 0 – 1023 significa (0% - 100%) en sentido anti horario CCW (0)
    b. Valores de 1024 – 2047 significa (0% - 100%) en sentido horario CW (1)
5. Voltaje actual: Es el valor del voltaje suministrado, este valor es 10 veces mayor el valor real. Por ejemplo, un valor de 100 representa 10 [V]
6. Temperatura actual: Es el valor de la temperatura interna expresada en grados Celsius. Por ejemplo, un valor de 30 representa 30 [°C]

Para el diseño del módulo en SIMULINK se optó por recibir los datos como los muestra la tabla 4.10.

Tabla 4.10 Cantidad de bytes que recibe SIMULINK módulo Dynamixel.

Parámetro	Rango de operación	Numero de bytes
ID	0 - 253	1 byte
Posición actual	0 - 300	2 bytes
Velocidad actual	0 - 1023	2 bytes
Carga actual	0 - 2047	2 bytes
Voltaje actual	0 - 190	1 byte
Temperatura actual	0 – 80	1 byte

Como se puede apreciar en la 4.10 se necesita recibir 9 bytes de información en formato uint8, es decir, los parámetros para el packet input son los siguientes: packet size = 9 y data type = ‘9*uint8’ como se muestra en la Imagen 4.16.

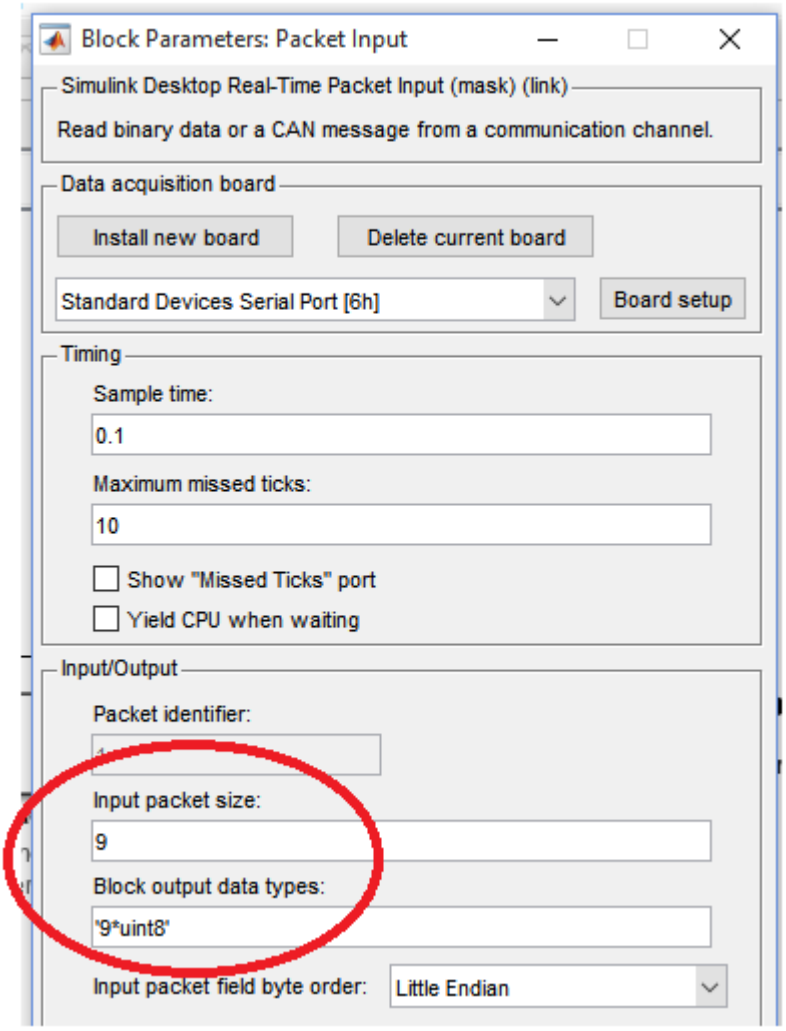


Imagen 4.16 Configuración de los bytes a recibir SIMULINK-Dynamixel.

Ya que se tienen identificados la cantidad de bytes a enviar y recibir en el módulo de SIMULINK (Arduino), es necesario crear un módulo que permita convertir un número mayor de 8 bits en dos números de 8 bits.

Para lo anterior haremos uso del bloque *Interpreted MATLAB Function*, este bloque permite utilizar un script (.m); dicho bloque indica el nombre del script, el número de entradas y el número de salidas que tendrá.

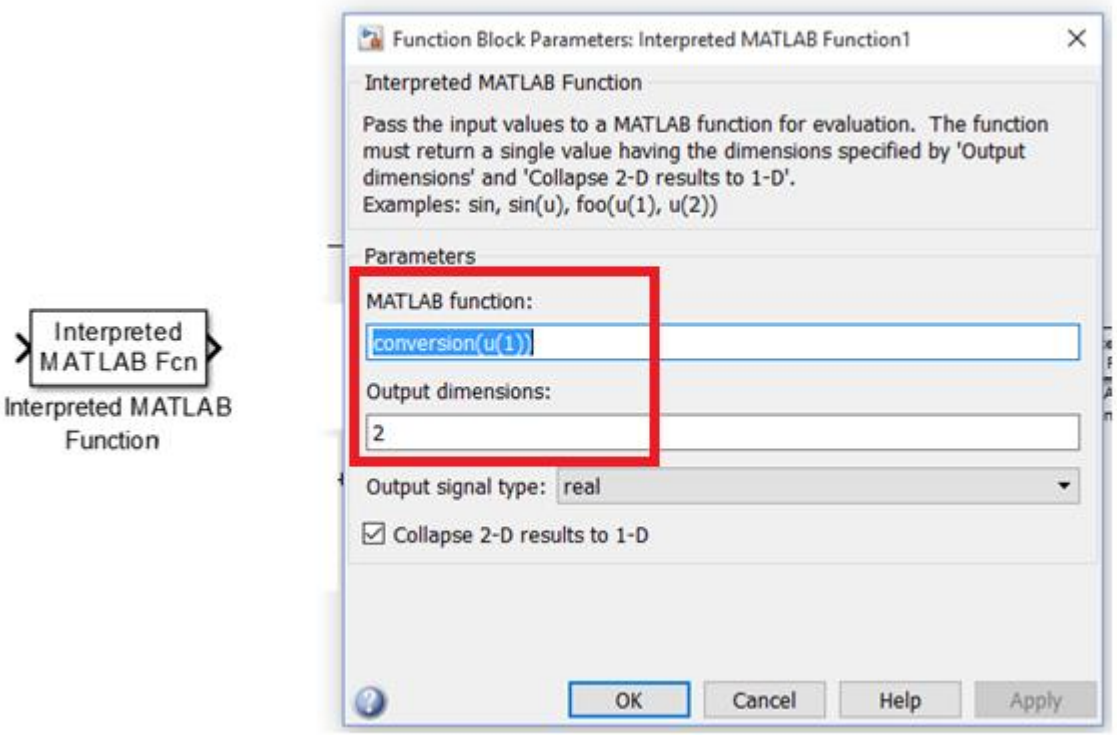


Imagen 4.17 Configuración del script conversión en el Interpreted MATLAB Function.

El script tiene algunas características como las que se muestran en la Tabla 4.11 para que funcione el bloque de *Interpreted MATLAB Function*. Una de las ventajas de usar este bloque es que además de permitir crear tus propias funciones en SIMULINK, también se puede ejecutar desde el Command Window de MATLAB, lo cual nos ayuda a probar que el script funciona adecuadamente antes de agregarlo al modelo de SIMULNK.

Tabla 4.11 Estructura de un script para agregarlo en un Interpreted MATLAB Function.

Descripción	Instrucciones MATLAB	Explicación
Indicar el comienzo del script	`function sal = conversion(u)`	conversión -> Nombre del script u -> Entrada sal -> Salida
Indicar que se hará con la entrada	`dat=uint16(u);`	La entrada se guarda en dat y se convierte en un entero sin signo de 16 bits

Acciones adicionales a realizar	`mas1=uint16(255);`
	
	`low=double(bitand(dat,mas1))`
	
	`high=double(bitand(swapbytes(dat),mas1));`
		Se hace una conversión del número 255 en un entero sin signo de 16 bits y se guarda en mas1 (mascara)
		
		Se realizan operaciones con dat y mas1. El resultado se almacena en high y low

Indicar la salida	`sal=[high; low];`	La salida es igual a una matriz 2 filas 1 columna con 2 datos (high - low), lo cual representa 2 salidas.

El script que se detalla en la Tabla 4.11 es el script que se creó para convertir un número mayor de 8 bits a 2 bytes de 8 bits en formato unit8, es decir, enteros sin signo de 8 bits, de esta manera se puede enviar información que sea mayor de 8 bits por el puerto serial (imagen 4.18).

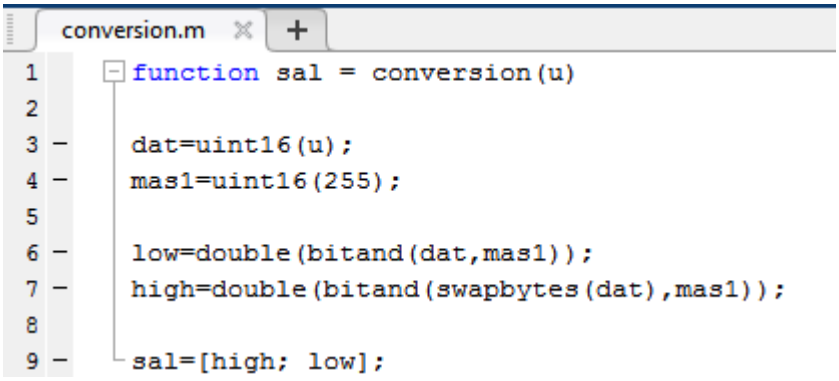


Imagen 4.18 Script conversión.

Si se desea convertir el número 299 a 2 enteros de 8 bits sin signo nos movemos a la carpeta en donde está guardado el script "conversión.m" e introducimos en el Command Window de MATLAB el nombre del script y dentro del paréntesis el número que se quiere convertir, es decir, *conversion(299)* y nos dará como resultado una matriz columna donde la primer fila representa el byte *high* y la segunda fila representa el byte *low* (imagen 4.19).

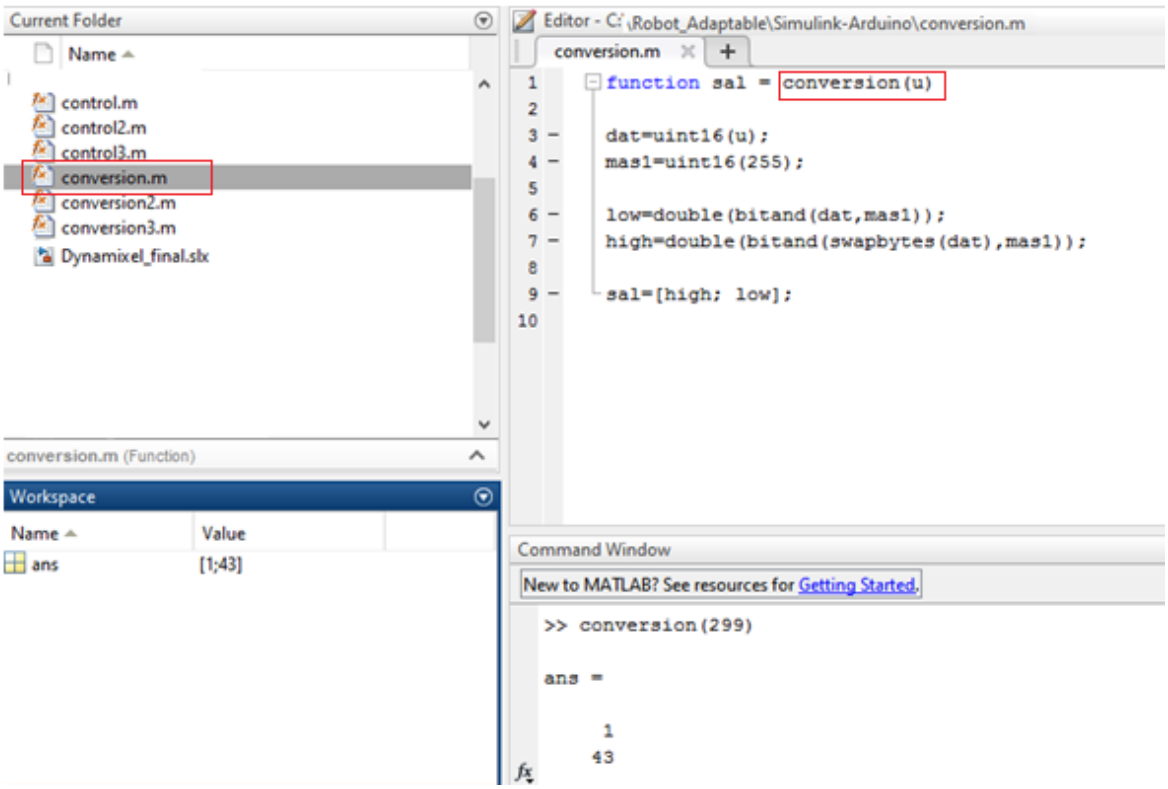


Imagen 4.19 Ejecución del script conversión desde el Command Window.

Ahora se requiere crear un script que convierta un número dividido en 2 bytes de 8 bits en un sólo número mayor de 8 bits. Dicho script tendrá dos datos, una entrada y una salida, y se llamará *conversion2*; sus entradas serán *high – low* y su salida será el dato que se obtiene de manipular los bytes *high* y *low*, quedando de la siguiente forma (imagen 4.20):

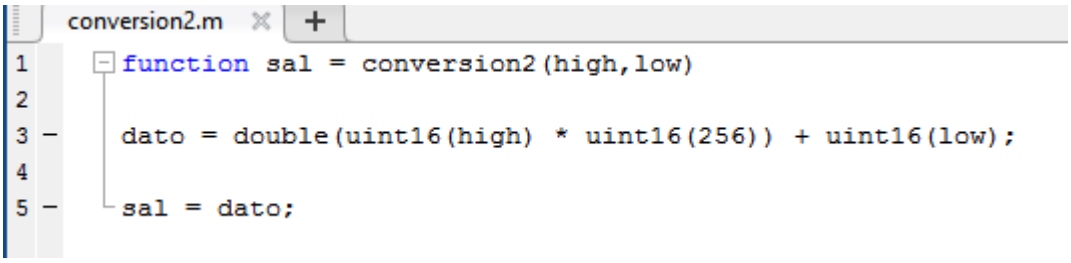


Imagen 4.20 Script conversión 2.

Continuando con el ejemplo anterior, en donde convertimos el número 299 en dos valores high = 1 y low = 43, si esos valores los introducimos en conversion2, debemos obtener como resultado el número 299. Para corroborar que el script funciona, hacemos un procedimiento análogo al que hicimos con conversion1. En el Command Windows de MATLAB introducimos conversion2 (1,43) (imagen 4.21).

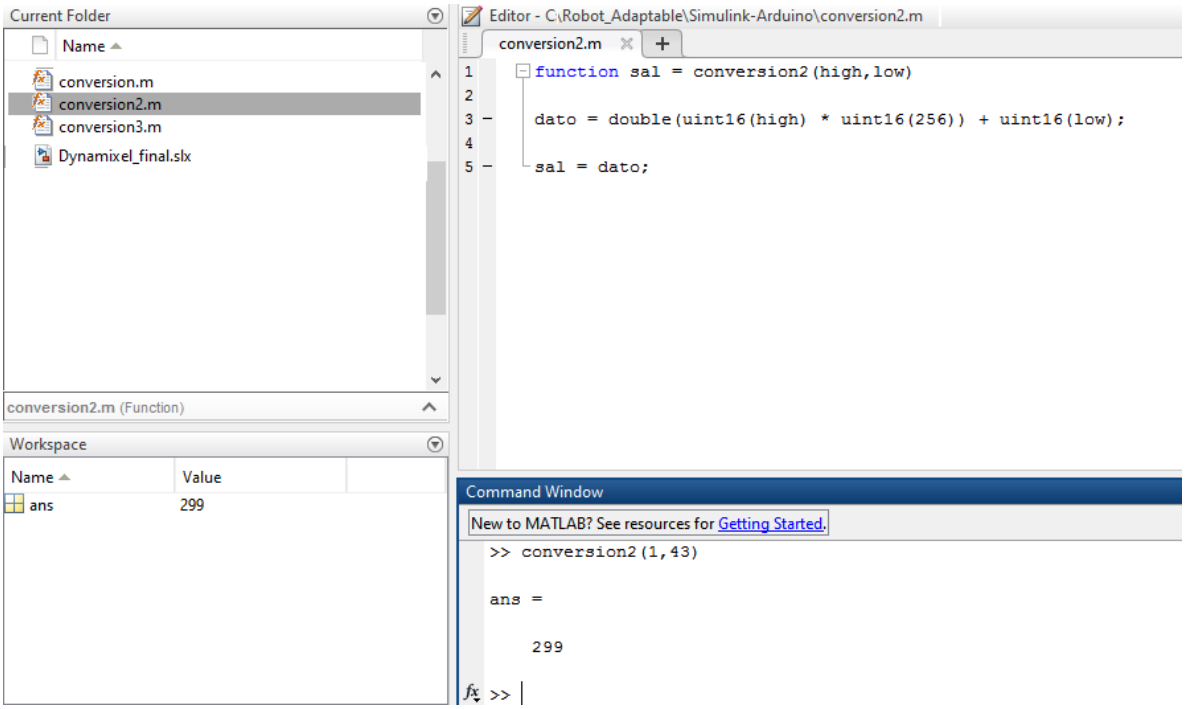


Imagen 4.21 Ejecución del script conversión2 desde el Command Window.

Para configurar el script conversión2 dentro del bloque de Interpreted MATLAB Function, se debe configurar las entradas y las salidas como se aprecia en la imagen 4.22:

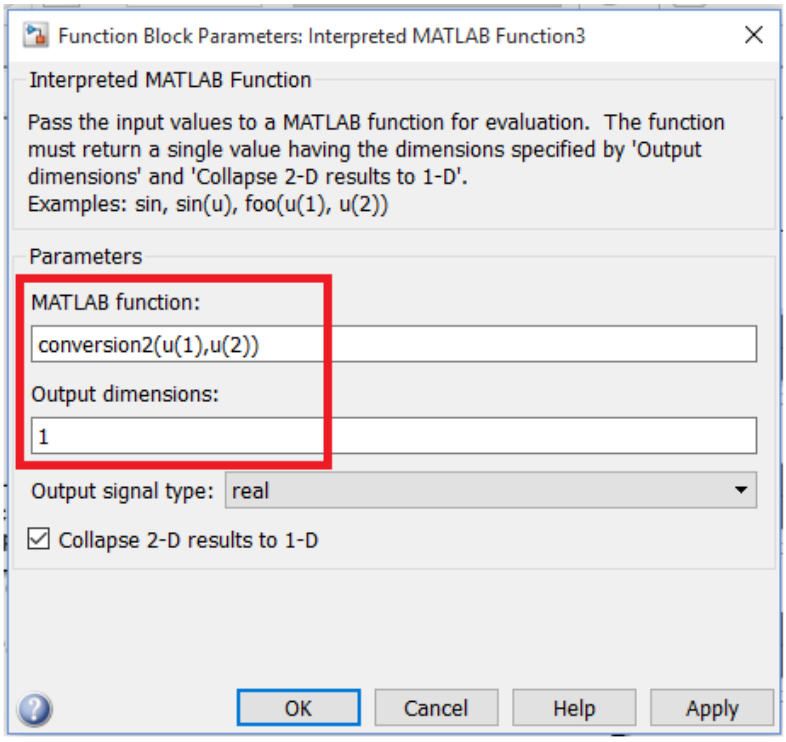


Imagen 4.22 Configuración del script conversión2 en el Interpreted MATLAB Function.

El script “conversion2” se usará para armar un sólo número mayor de 8 bits las variables de posición deseada, velocidad y carga, como se había mencionado anteriormente, llegan en dos bytes de 8 bits.

Posteriormente, se requiere un post-procesamiento de la información de algunas variables como la Velocidad, la Carga y el Voltaje, por lo cual se realizaron 3 scripts como se describen a continuación.

El primer script de post-procesamiento es el de Velocidad, el cual permite obtener la velocidad en RPM, como se había mencionado es necesario multiplicar el número obtenido de la lectura de velocidad de los Dynamixel (0 – 1023) por 0.111, el resultado será la Velocidad en RPM (imagen 4.23).

velocidad.m

```
1   function sal = velocidad(dato)
2
3 -   velocidad = dato * 0.111;
4
5 -   sal = velocidad;
```

Imagen 4.23 Script velocidad

Haciendo la prueba del script usaremos el dato mayor que se puede obtener de las lecturas de Velocidad de los Dynamixel que es 1023, lo que equivale aproximadamente a 113.5 RPM y se puede comprobar desde el Command Window de MATLAB poniendo el nombre del script y entre paréntesis el valor deseado, como lo muestra la imagen 4.24.

Command Window

New to MATLAB? See resources for Getting Started.

```
>> velocidad(1023)

ans =

  113.5530

>> velocidad(250)

ans =

   27.7500
```

Imagen 4.24 Ejecución del script velocidad desde el Command Window.

De esta forma verificamos que el post-procesamiento de la información de la Velocidad funciona de manera adecuada y podemos ingresarlo a un *Interpreted MATLAB Function* que tenga una entrada y una salida (imagen 4.25).

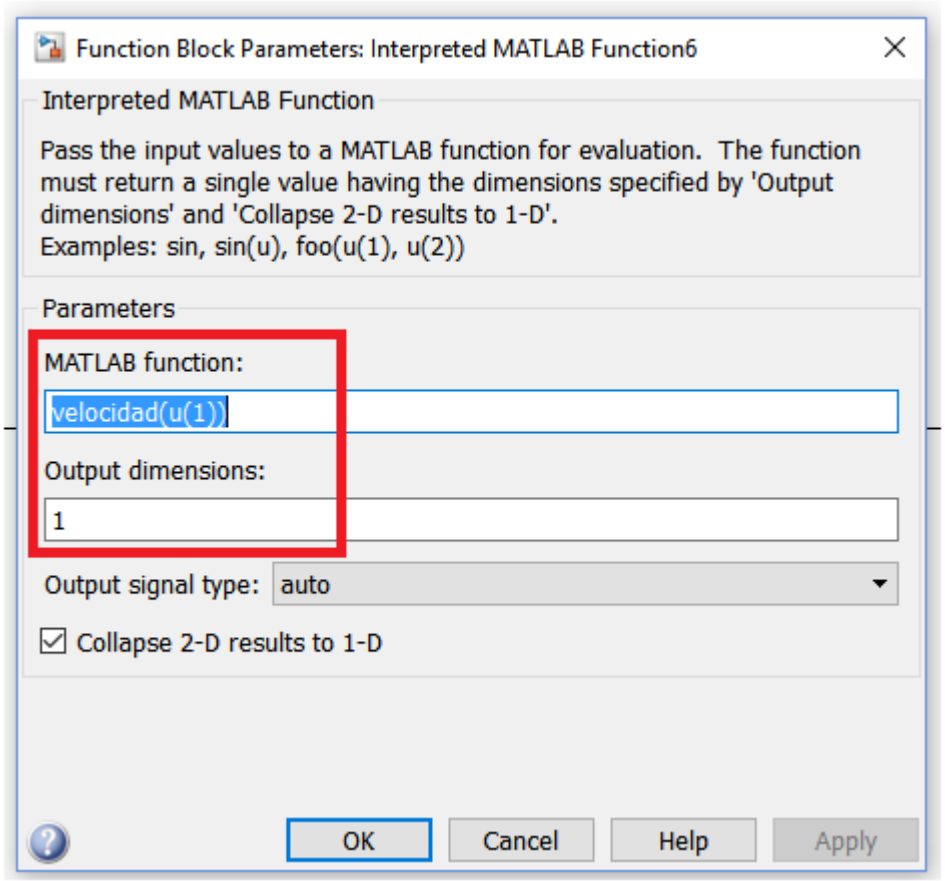


Imagen 4.25 Configuración del script velocidad en el Interpreted MATLAB Function.

El segundo script de post-procesamiento es el de Carga, el cual permite obtener dos valores el primero es porcentaje de carga y el segundo el sentido de giro. El valor obtenido en dicha lectura puede estar entre 0 y 2047 y se descompone de la siguiente manera:

1. El porcentaje de carga (0% - 100%)
2. El sentido en que la carga está trabajando (Sentido horario [0], sentido anti horario [1])
   a. Valores de 0 – 1023 significa (0% - 100%) en sentido anti horario CCW
   b. Valores de 1024 – 2047 significa (0% - 100%) en sentido horario CW.

Un valor de 1023 representa el 100% de carga en sentido anti horario [1], un valor de 1536 representa el 50 % de carga en sentido horario [0] (imagen 4.26).

carga.m

```
function sal = carga(dato)

if dato <= 1023
    dato  = (dato*100)/1023 ;
    sentido = 1;
end

if dato > 1023
    dato = dato - 1024;
    dato = (dato*100)/1023 ;
    sentido = 0;
end

sal = [dato; sentido];
```

Imagen 4.26 Script carga

Para hacer la prueba del script se utilizaron los siguientes valores 512, 1023 y 1536 en el Command Window y se obtuvieron los valores mostrados en la imagen 4.27:

Command Window

New to MATLAB? See resources for Getting Started.

```
>> carga(512)

ans =

   50.0489
    1.0000

>> carga(1536)

ans =

   50.0489
         0

>> carga(1023)

ans =

   100
     1
```

Imagen 4.27 Ejecución del script carga desde el Command Window.

De esta forma verificamos que el post-procesamiento de la información de la Carga funciona de manera adecuada y podemos ingresarlo a un *Interpreted MATLAB Function* que tenga una entrada y dos salidas, como lo muestra la imagen 4.28.

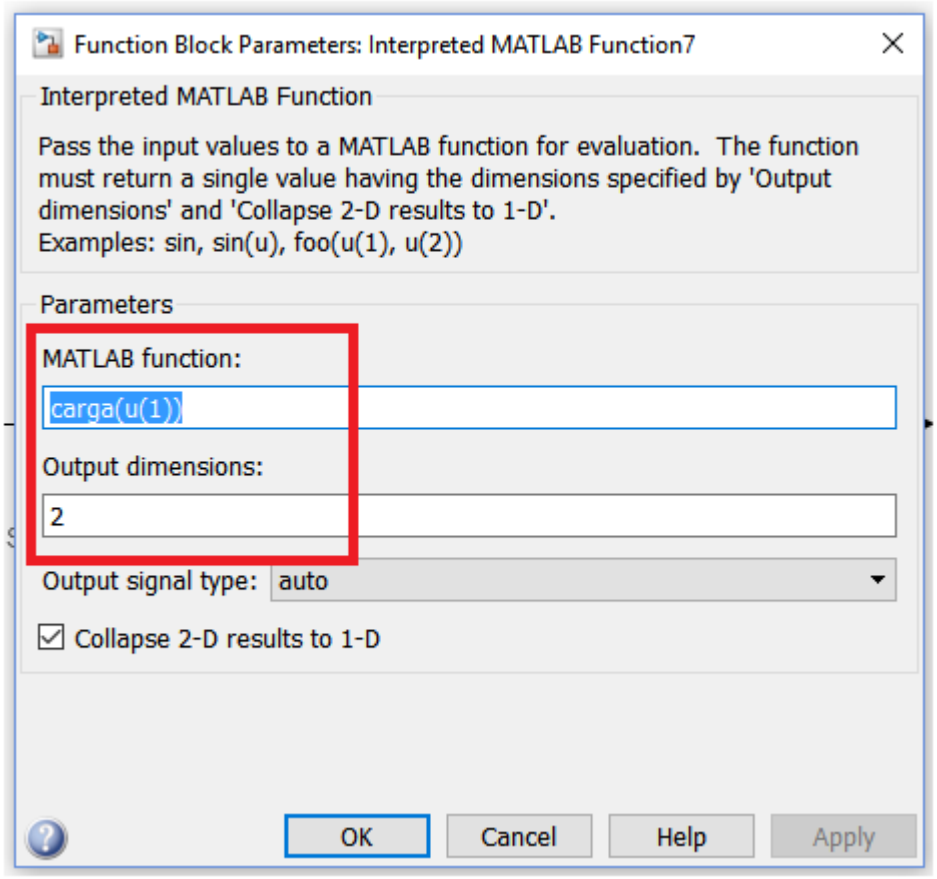


Imagen 4.28 Configuración del script carga en el Interpreted MATLAB Function.

El tercer script de post-procesamieto es el Voltaje, el cual se obtiene diviendo entre 10 la lectura que se obtiene de los Dynamixel. El tener 10 veces el valor real como lectura se evita manejar números con punto decimal al momento de enviarse estos por comunicación serial y se maneja como entero sin signo de 8 bits haciendo que sea más fácil él envió de la información por este protocolo de comunicación y el post-procesamiento es sencillo (imagen 4.29).

voltaje.m

```
1  function sal = voltaje(dato)
2
3  dato = dato/10;
4
5  sal = dato;
```

Imagen 4.29 Script voltaje

Para probar el script se pusieron los siguientes valores 123 y 129 en el Command Window, obteniendo los valores que se muestran en la imagen 4.30.

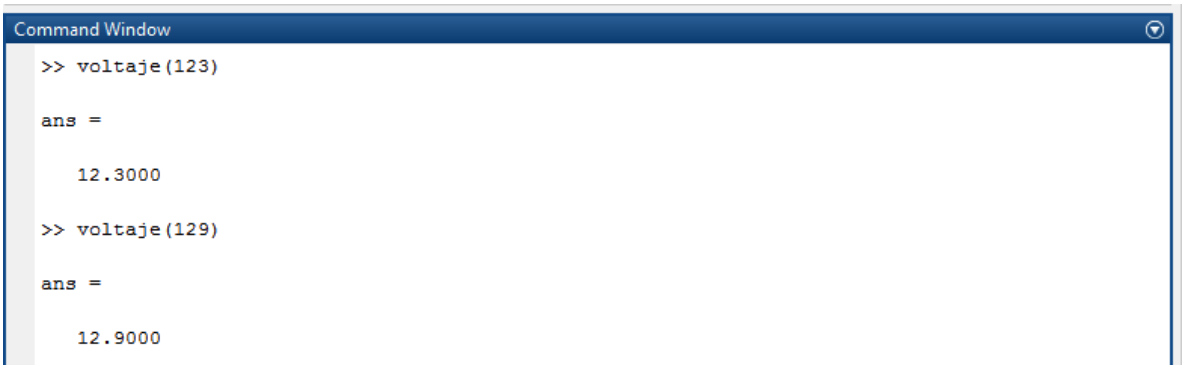


Imagen 4.30 Ejecución del script voltaje desde el Command Window.

De esta forma, se comprueba que el post-procesamiento de la información del Voltaje funciona de manera adecuada y es posible ingresarlo a un *Interpreted MATLAB Function* que tenga una entrada y una salida (imagen 4.31).

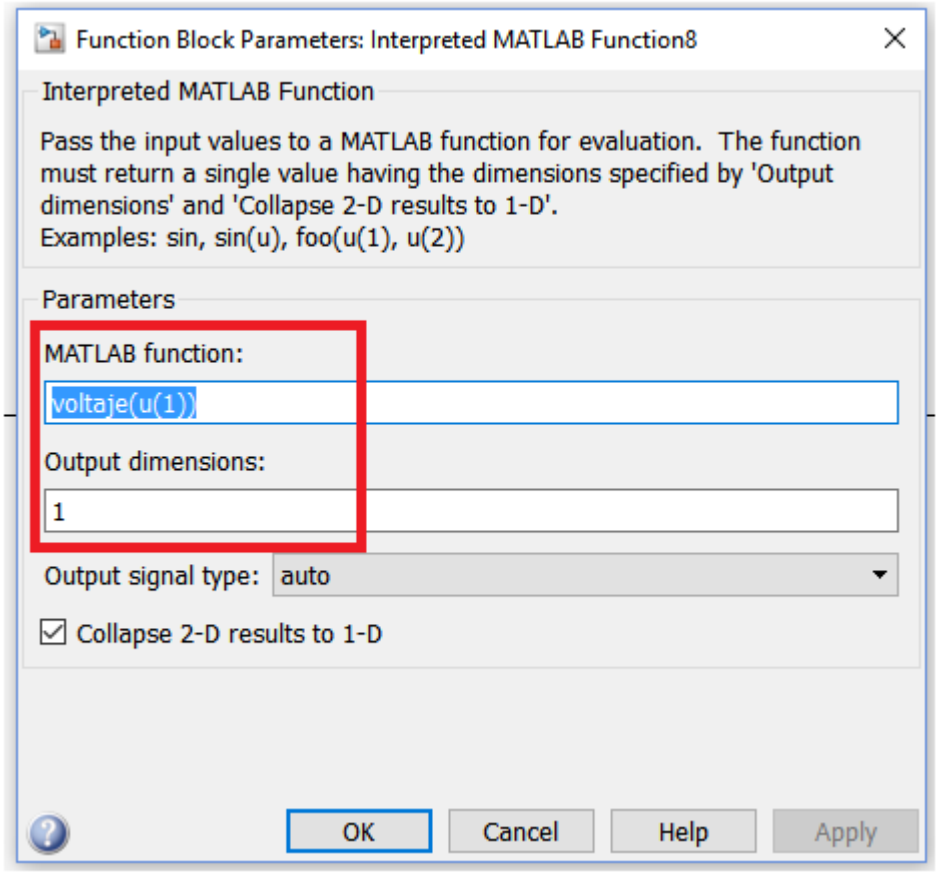


Imagen 4.31 Configuración del script voltaje en el Interpreted MATLAB Function.

Finalmente se creó el módulo SIIMULINK (Dynamixel), el cual tiene 3 entradas y 6 salidas como se había establecido en las especificaciones, quedando como lo muestra la imagen 4.32.

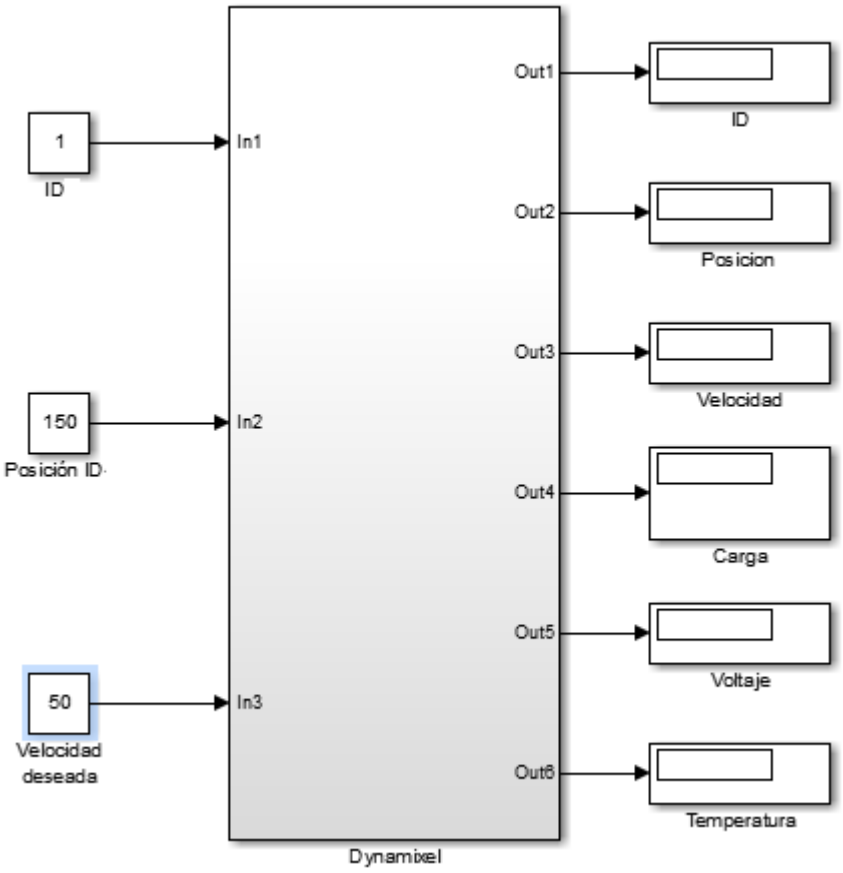


Imagen 4.32 Modulo SIMULINK - Dynamixel.

El módulo Dynamixel internamente está estructurado de la siguiente manera (imagen 4.33):

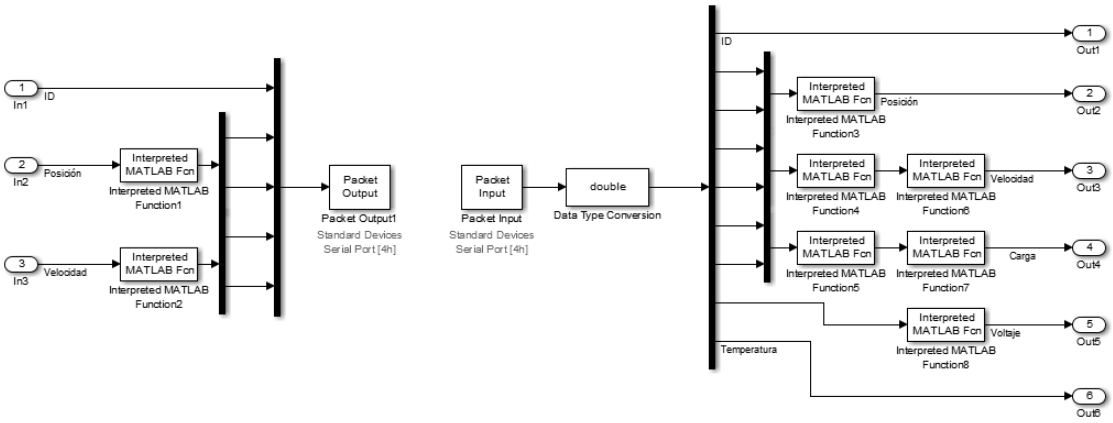


Imagen 4.33 Estructura interna del módulo SIMULINK – Dynamixel.

El microcontrolador, es el encargado de recibir los datos que se envían desde SIMULINK, el que controla y lee las variables de interés de los servomotores Dynamixel y el que manda la información leída de los Dynamixel a SIMULINK. Para lo anterior se creó el siguiente algoritmo (imagen 4.34):

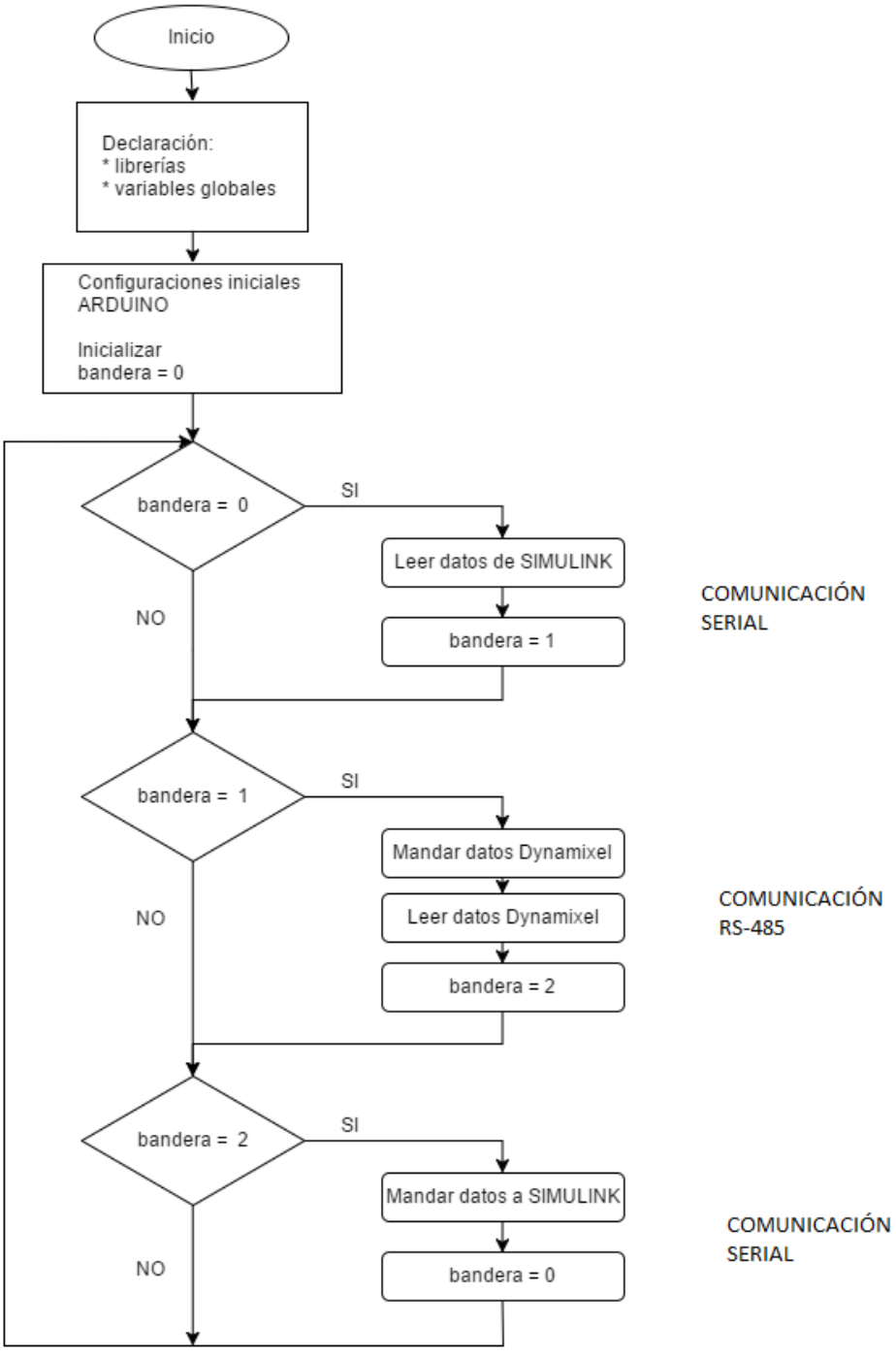


Imagen 4.34 Algoritmo para Arduino (módulo SIMULINK – Dynamixel).

Éste algoritmo se programó en el microcontrolador Arduino y se puede consultar el programa completo en el Apéndice 9 (Controlador SIMULINK – Dynamixel). Como se mencionó anteriormente, se hará uso de la librería “DynamixelSerial1.h”, para utilizar algunas de las instrucciones que están disponibles. Si se desea utilizar instrucciones adicionales a las empleadas en éste proyecto, es recomendable consultar el Apéndice 4.

A continuación, se detallan parámetros importantes para realizar de manera adecuada la comunicación que se estableció desde SIMULINK con Arduino. Se recuerda que desde SIMULINK se enviaban 5 bytes: 1) ID, 2) Posición deseada (high), 3) Posición deseada (low), 4) Velocidad deseada (high) y 5) Velocidad deseada (low). Para leer dichos bytes en Arduino se realiza de la siguiente manera (imagen 4.35):

```
byte  id, posd_h, posd_l, vel_h, vel_l;

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

void loop ()
{
   if(Serial.available()>0)
   {
     id = Serial.read();
     posd_h = Serial.read()
     posd_l = Serial.read()
     vel_h = Serial.read();
     vel_l = Serial.read();
   }
 }
```

Imagen 4.35 Leer los 5 bytes que envía SIMULINK para los Dynamixel en Arduino.

Como en SIMULINK fue necesario separar números mayores a 8 bits en 2 bytes para poder enviar dicha información por el puerto serial, en Arduino se requirió hacer el procedimiento inverso, es decir, hacer un número mayor de 8 bits desde los 2 bytes que se leyeron (high, low). Esto se realiza de la siguiente manera (imagen 4.36):

```
int   posd, vel;

posd = posd_h * 256 + posd_l;
vel = vel_h * 256 + vel_l;
```

Imagen 4.36 Armar número mayor a 8 bits con 2 bytes en Arduino.

Ya que se tienen los 3 datos de interés, ID, posd (posición deseada) y vel (velocidad deseada) se enviará la información a los servomotores Dynamixel. Un parámetro importante para la

correcta comunicación con los Dynamixel es establecer la velocidad de transmisión (Baudrate) en 1 Mbps. Para configurar tanto el ID y el Baudrate de los Dynamixel consultar el Apéndice 7. Recordando que el rango de operación de la posición de los Dynamixel RX es de (0° - 300°) se necesita enviarle valores de (0 – 1024) se hace una conversión en Arduino para poder enviarle la información a los Dynamixel, lo anterior se logra de la siguiente manera (imagen 4.37):

```
#include <DynamixelSerial1.h>

void setup ()
{
  Serial.begin(9600);
  Dynamixel.begin(1000000,2);     // Inicialize the servo at 1Mbps and Pin Control 2
}

void loop ()
{
   .
   .
   .
   int posd_val = map(posd,0,300,0,1024);
   Dynamixel.moveSpeed (id,posd_val,vel);
}
```

Imagen 4.37 Conversión de datos de una escala a otra escala en Arduino.

Para la lectura de las variables de interés de los Dynamixel: 1) ID (ya se tiene), 2) Posición actual, 3) Velocidad actual, 4) Carga, 5) Voltaje, 6) Temperatura se realiza como semuestra en la imagen 4.38.

```
int   posa, posa_val, Speed, Load, Voltage, Temperature;

Speed = Dynamixel.readSpeed(id);
posa_val = Dynamixel.readPosition(id);
Load = Dynamixel.readLoad(id);
Voltage = Dynamixel.readVoltage(id);
Temperature = Dynamixel.readTemperature(id);
```

Imagen 4.38 Lectura de las variables de Dynamixel en Arduino.

Al realizar las lecturas, se obtiene algunas variables de números enteros mayores a 8 bits, por lo cual es necesario separar esas variables en 2 bytes (high y low), de manera análoga a como se hizo en SIMULINK para poder enviar la información por el puerto serial; además, en el caso de la posición actual se hará una conversión de los valores (0 – 1023) que se leen con la instrucción Dynamixel.readPosition(id) en valores de (0° - 300°). La manera de hacer esto en Arduino es como lo muestra la imagen 4.39.

```
int   posa, posa_val, Speed, Load, Voltage, Temperature;
byte  posa_h, posa_l, Speed_h, Speed_l, Load_h, Load_l;

posa = map(posa_val,0,1024,0,300);
posa_h = posa>>8;
posa_l = posa;
Speed_h = Speed>>8;
Speed_l = Speed;
Load_h = Load>>8;
Load_l = Load;
```

Imagen 4.39 Separar un número mayor de 8 bits en 2 bytes en Arduino.

Finalmente, la manera de enviar los 9 bytes a SIMULINK desde Arduino es la siguiente:

```
Serial.write(id);
Serial.write(posa_h);
Serial.write(posa_l);
Serial.write(Speed_h);
Serial.write(Speed_l);
Serial.write(Load_h);
Serial.write(Load_l);
Serial.write(Voltage);
Serial.write(Temperature);
```

Imagen 4.40 Enviar los 9 bytes de las lecturas de los Dynamixel a SIMULINK desde Arduino.

De ésta manera es como finaliza el diseño del módulo en SIMULINK que permita enviar y recibir datos de los servomotores Dynamixel, cumpliendo con las especificaciones que se establecieron en el Paso 1 del Diseño de software.

4.2.5 Diseño del módulo SIMULINK (MD25)

Para la Tarjeta MD25 se tiene una tabla de control, la cual contiene la información de los registros que se pueden leer o escribir, dicha tabla nos indica cuales las variables que se pueden leer y cuáles son los registros de configuración.

En las especificaciones se determinaron como parámetros importantes para enviar a la Tarjeta MD25: 1) Dirección de la Tarjeta, 2) Velocidad del motor A, 3) Velocidad del motor B. Estos parámetros se encuentran en la tabla de control 4.12 y se puede consultar de qué manera operan.

Tabla 4.12. Tabla de control Tarjeta MD25. Selección de los parámetros para enviar.

Register	Name	Read/Write	Description
0	Speed1	R/W	Motor1 speed (mode 0,1) or speed (mode 2,3)
1	Speed2/Turn	R/W	Motor2 speed (mode 0,1) or turn (mode 2,3)
2	Enc1a	Read only	Encoder 1 position, 1st byte (highest), capture count when read
3	Enc1b	Read only	Encoder 1 position, 2nd byte
4	Enc1c	Read only	Encoder 1 position, 3rd byte
5	Enc1d	Read only	Encoder 1 position, 4th (lowest byte)
6	Enc2a	Read only	Encoder 2 position, 1st byte (highest), capture count when read
7	Enc2b	Read only	Encoder 2 position, 2nd byte
8	Enc2c	Read only	Encoder 2 position, 3rd byte
9	Enc2d	Read only	Encoder 2 position, 4th byte (lowest byte)
10	Battery volts	Read only	The supply battery voltage
11	Motor 1 current	Read only	The current through motor 1
12	Motor 2 current	Read only	The current through motor 2
13	Software Revision	Read only	Software Revision Number
14	Acceleration rate	R/W	Optional Acceleration register
15	Mode	R/W	Mode of operation (see below)
16	Command	Write only	Used for reset of encoder counts and module address changes

1. Dirección de la tarjeta MD25: Indica que Tarjeta MD25 queremos manipular. Se pueden configurar hasta 8 direcciones. Como se muestra en la tabla 4.11.
2. Velocidad del motor A: Se puede enviar valores de (0 – 255). En donde 0 representa la velocidad máxima hacia atrás, 128 representa stop, y 255 representa la velocidad máxima hacia adelante.
3. Velocidad del motor B: Se pueden enviar valores de (0 – 255). Y su funcionamiento es análogo a la velocidad del motor A.

Para poder cambiar la dirección de comunicación I2C de la tarjeta MD25 es necesario escribir una secuencia de comandos en el orden correcto seguido de la dirección que se le quiere asignar. La secuencia de datos es la siguiente (0xA0, AxAA, 0XA5, Address). Las posibles direcciones que se pueden asignar van desde B0 hasta BE, como se muestra en la tabla 4.13.

Tabla 4.13. Identificación de la dirección de la MD25 con el número de destellos.

Address Hex	Long Flash	Short Flashes
B0	1	0
B2	1	1
B4	1	2
B6	1	3
B8	1	4
BA	1	5
BC	1	6
BE	1	7

Por default las tarjetas MD25 vienen configuradas como B0, una vez que se haya cambiado la dirección es recomendable etiquetar la tarjeta, sin embargo, si se olvida la dirección de la tarjeta se puede identificar alimentando la tarjeta (sin enviar ningún comando) y ver como parpadea el LED verde. Un destello largo seguido de una serie de destellos cortos los cuales indican su dirección.

El programa para cambiar la dirección de comunicación I^2C de las tarjetas MD25 se encuentra en el Apéndice 8. Si no es posible identificar la dirección de la tarjeta con la cantidad de destellos, también se encuentra en el mismo apéndice, es un programa que está basado en el I^2CScanner (PROMETEC, 2016), el cual identifica la dirección de comunicación de algún dispositivo conectado por protocolo I^2C y además regresa al valor por default B0 de la tarjeta MD25.

Se detectó que para establecer comunicación con la tarjeta (default) la dirección de comunicación I^2C es el 0x58 y si se cambia la dirección como se explicó anteriormente, se encuentra la relación de la tabla 4.14:

Tabla 4.14. Dirección I2C dependiendo el último dato que se le envía.

Dato	Address I^2C	
	HEX	DEC
B0	0x58	88
B2	0x59	89
B4	0x5A	90

B6	0x5B	91
B8	0x5C	92
BA	0x5D	93
BC	0x5E	94
BE	0x5F	95

Para el diseño del módulo en SIMULINK se determinó enviar los datos como lo muestra la tabla 4.15.

Tabla 4.15. Cantidad de bytes a enviar desde SIMULINK (Módulo MD25).

Parámetro	Rango de operación	Numero de bytes
Dirección de la tarjeta MD25	88 - 95	1 byte
Velocidad motor A	0 - 255	1 byte
Velocidad motor B	0 - 255	1 byte

Como se puede apreciar en la Tabla 4.15, se requiere enviar 3 bytes de información en formato uint8, es decir, los parámetros para el packet output son los siguientes: packet size = 3 y data type = ‘3*uint8’ como se muestra en la Imagen 4.41.

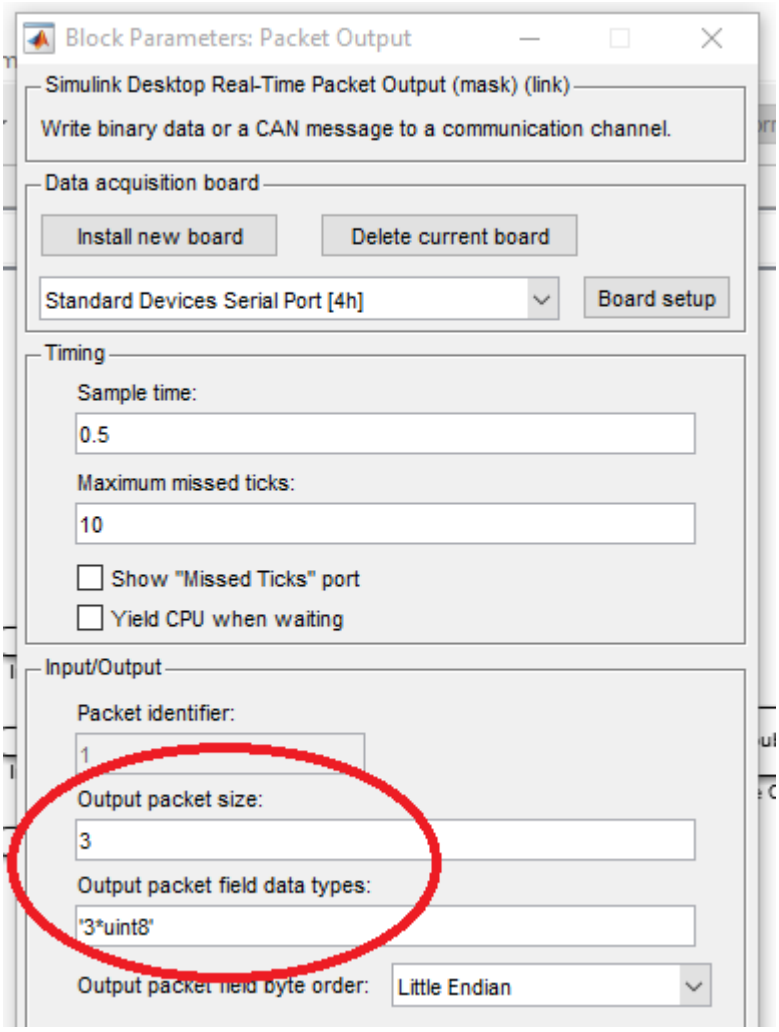


Imagen 4.41. Configuración de los bytes de envió SIMULINK-MD25.

En las especificaciones se determinaron como parámetros importantes para leer de las Tarjetas MD25: 1) Dirección de la Tarjeta MD25, 2) Encoder acoplado al motor A, 3) Encoder acoplado al motor B, 4) Voltaje, 5) Corriente motor A, 6) Corriente motor B. Estos parámetros se encuentran en la tabla de control y se consulta el funcionamiento de cada uno de ellos como lo muestra la tabla 4.16.

Tabla 4.16. Tabla de control Tarjeta MD25. Selección de los parámetros para recibir.

Register	Name	Read/Write	Description
0	Speed1	R/W	Motor1 speed (mode 0,1) or speed (mode 2,3)
1	Speed2/Turn	R/W	Motor2 speed (mode 0,1) or turn (mode 2,3)
2	Enc1a	Read only	Encoder 1 position, 1st byte (highest), capture count when read
3	Enc1b	Read only	Encoder 1 position, 2nd byte
4	Enc1c	Read only	Encoder 1 position, 3rd byte
5	Enc1d	Read only	Encoder 1 position, 4th (lowest byte)
6	Enc2a	Read only	Encoder 2 position, 1st byte (highest), capture count when read
7	Enc2b	Read only	Encoder 2 position, 2nd byte
8	Enc2c	Read only	Encoder 2 position, 3rd byte
9	Enc2d	Read only	Encoder 2 position, 4th byte (lowest byte)
10	Battery volts	Read only	The supply battery voltage
11	Motor 1 current	Read only	The current through motor 1
12	Motor 2 current	Read only	The current through motor 2
13	Software Revision	Read only	Software Revision Number
14	Acceleration rate	R/W	Optional Acceleration register
15	Mode	R/W	Mode of operation (see below)
16	Command	Write only	Used for reset of encoder counts and module address changes

1. Dirección tarjeta MD25: Indica de que tarjeta se está leyendo la información. (88 – 95).
2. Encoder acoplado al motor *A*: Tiene acoplado un encoder, los cuales tienen asociado un conjunto de 4 bytes de 8 bits cada uno, generando un número de 32 bits con signo.
3. Encoder acoplado al motor *B*: Funciona de igual manera que el encoder del motor *A*.
4. Voltaje: Es el valor del voltaje de operación de los motores, este valor es 10 veces mayor al valor real. Por ejemplo, un valor de 121 representa 12.1 V.
5. Corriente motor A: Es una lectura de la corriente a través del motor A, este valor es 10 veces el valor real, Por ejemplo, un valor de 19 representa 1.9 A.
6. Corriente motor *B*. Funciona de igual manera que la corriente del motor *A*.

Para el diseño del módulo en SIMULINK de la MD25 se estableció recibir los datos de la siguiente manera (tabla 4.17):

Tabla 4.17. Cantidad de bytes que recibe SIMULINK módulo MD25

Parámetro	Rango de operación	Numero de bytes
Dirección tarjeta MD25	88 - 95	1 byte
Encoder motor A	-2^{31} - 2^{31}	4 bytes

Encoder motor B	-2^{31} - 2^{31}	4 bytes
Voltaje	0 - 140	1 bytes
Corriente motor A	0 - 25	1 byte
Corriente motor B	0 – 25	1 byte

Como se puede apreciar en la tabla 4.17, se necesita recibir 12 bytes de información, sin embargo, los registros de los encoders son de 32 bits con signo, y los demás registros son enteros de 8 bits sin signo, por lo cual, se tiene un problema al definir el tipo de dato que se recibirá. Para resolver este problema se hará un script (.m) en donde se convierta los 4 registros uint8 en un int32. A este script se nombró conversión3.m. (Imagen 4.43).

conversion3.m

```
function sal = conversion3(b3,b2,b1,b0)

dato = (uint32(b3) * uint32(256^3)) + (uint32(b2) * uint32(256^2)) + (uint32(b1) * uint32(256)) + uint32(b0);
dato = double(dato);

if dato > 2^31
    dato = dato - 2^32;
end

sal = dato;
```

Imagen 4.43. Script conversion3.

Para probar el funcionamiento de “conversion3.m” desde el Command Window de MATLAB se introdujo el primer caso, cuando a los registros les llega 255, 255, 255, 255 se debe interpretar como un -1; el menos -2 le debe llegar a los registros la siguiente combinación de bytes 255, 255, 255, 254. Ambos casos se probaron desde el Command Windows obteniendo los resultados deseados como se aprecia en la Imagen 4.44.

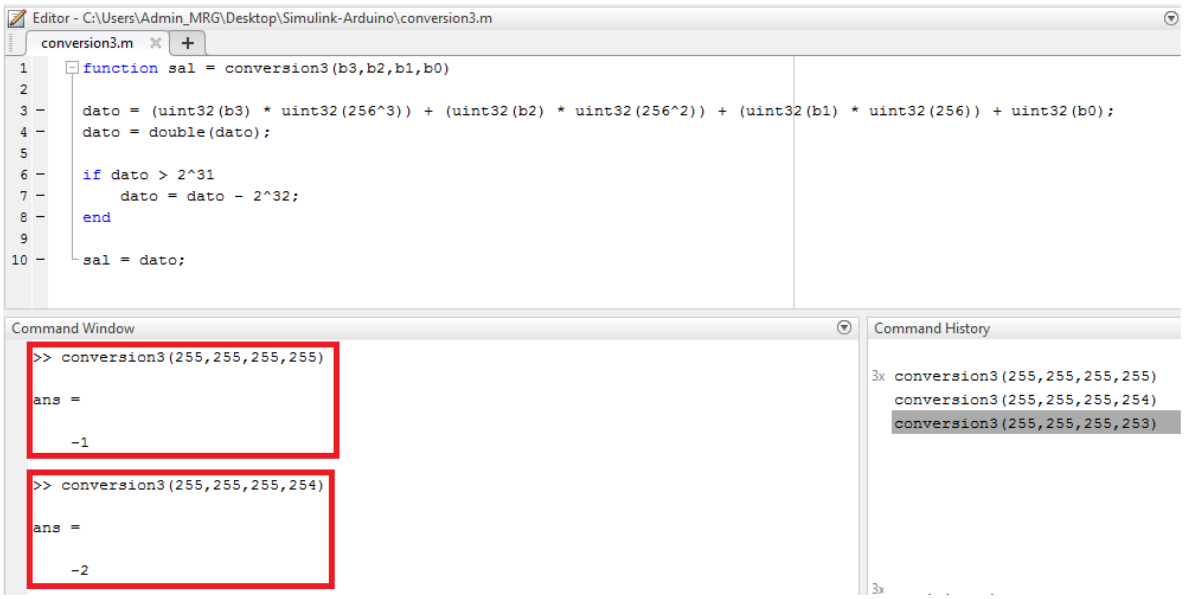


Imagen 4.44. Ejecución del script conversión3 desde el Command Window.

Las configuraciones para usar conversion3.m en el Interpreted MATLAB Function son 4 entradas y una salida (imagen 4.45).

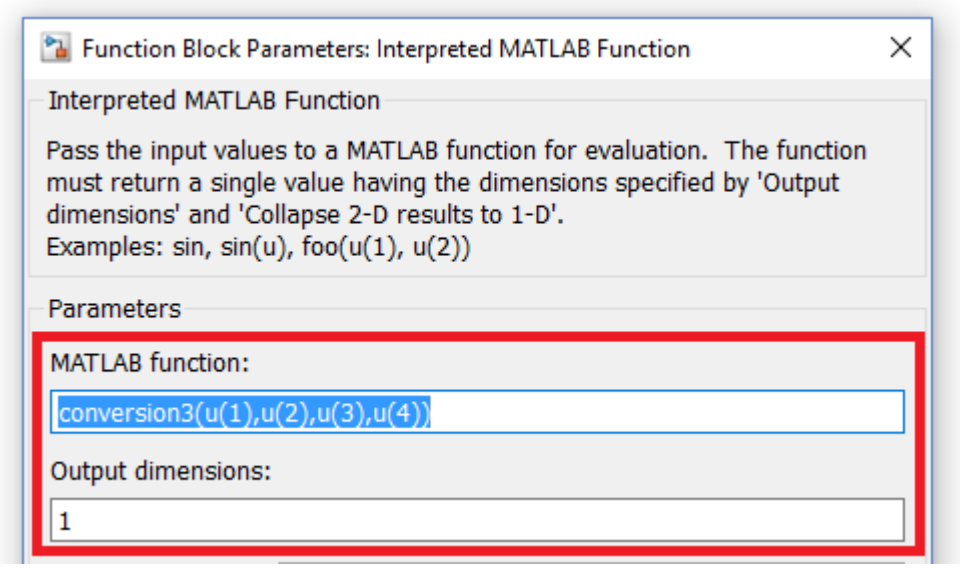


Imagen 4.45. Configuración del script conversión3 en el Interpreted MATLAB Function.

De esta manera se resuelve el problema con el tipo de datos que se recibirá en el Packet Input y los parámetros para dicho bloque son los siguientes: packet size = 12 y data type = '12*uint8' como se muestra en la Imagen 4.46.

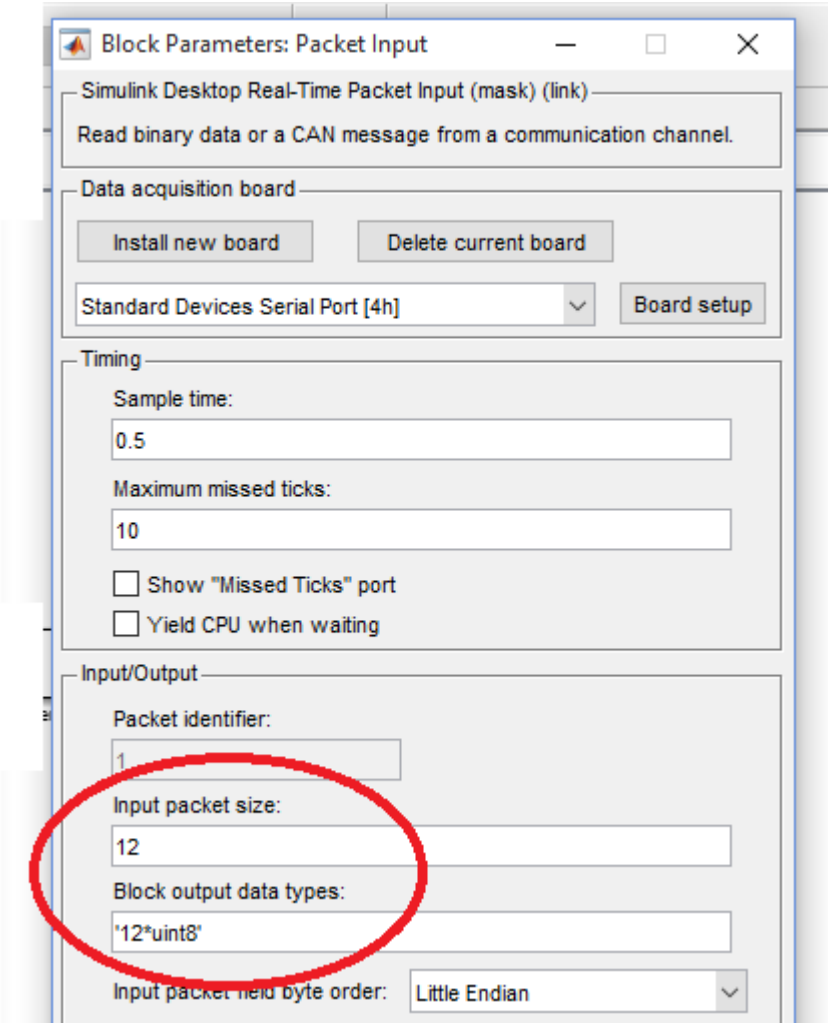


Imagen 4.46. Configuración de los bytes a recibir SIMULINK-MD25.

Finalmente se creó el módulo en SIMULINK (MD25), el cual tiene 3 entradas y 6 salida como se había establecido en las especificaciones, quedando como lo muestra la imagen 4.47.

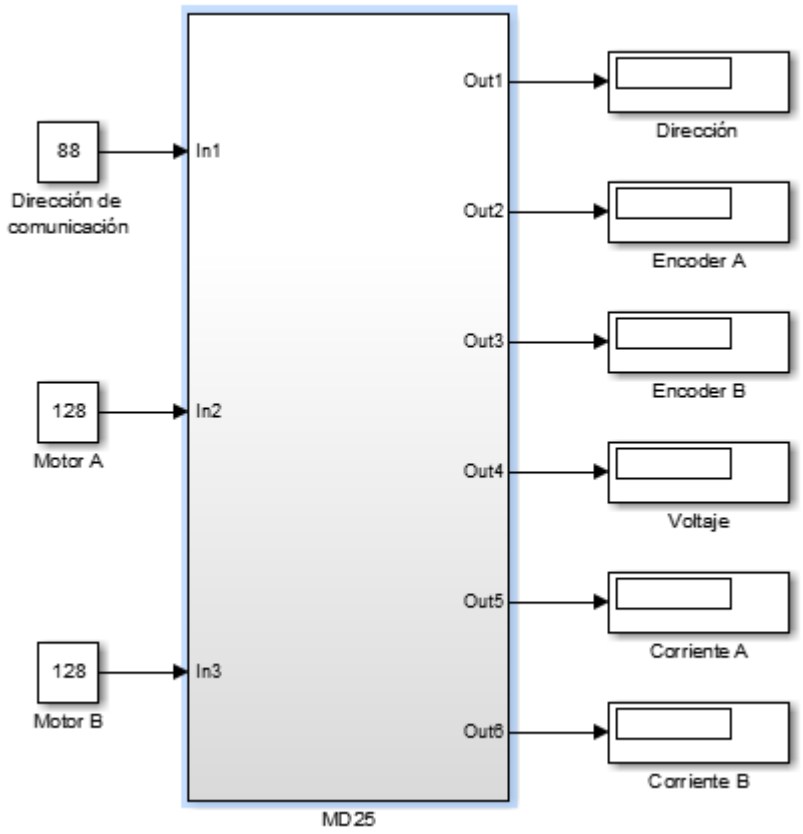


Imagen 4.47. Módulo SIMULINK – MD25.

El módulo MD25 internamente está estructurado de la siguiente manera (imagen 4.48):

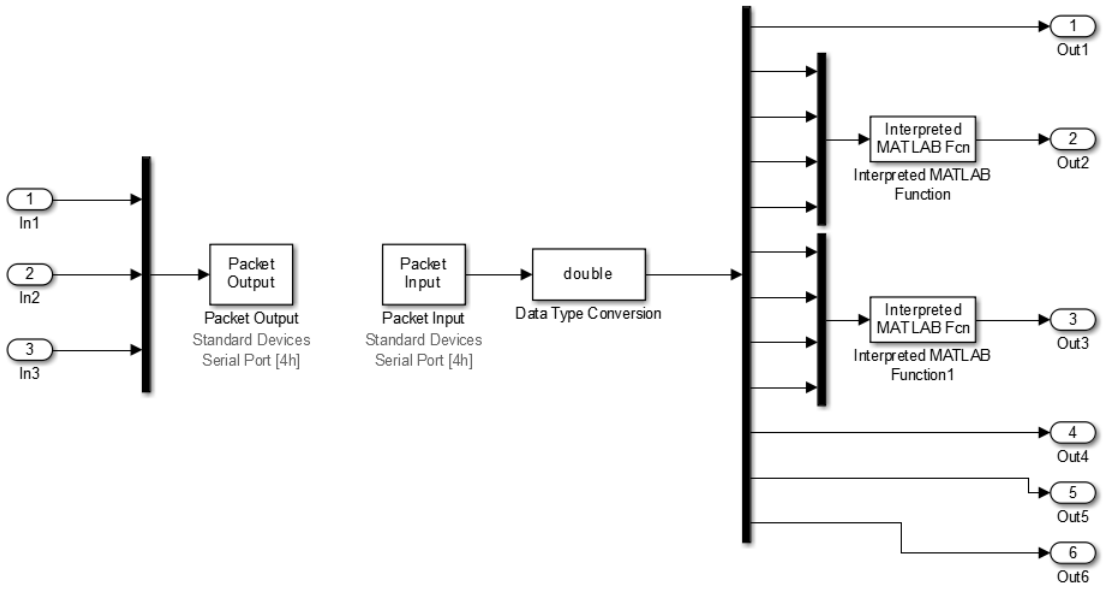


Imagen 4.48. Estructura interna del Módulo SIMULINK – MD25.

En el caso del microcontrolador, el cual es el encargado de recibir los datos que se envían desde SIMULINK, el que envía la información a la Tarjeta MD25, el que recibe los datos de interés y el que le envía la información a SIMULINK. Para lo anterior se creó el siguiente algoritmo (imagen 4.49):

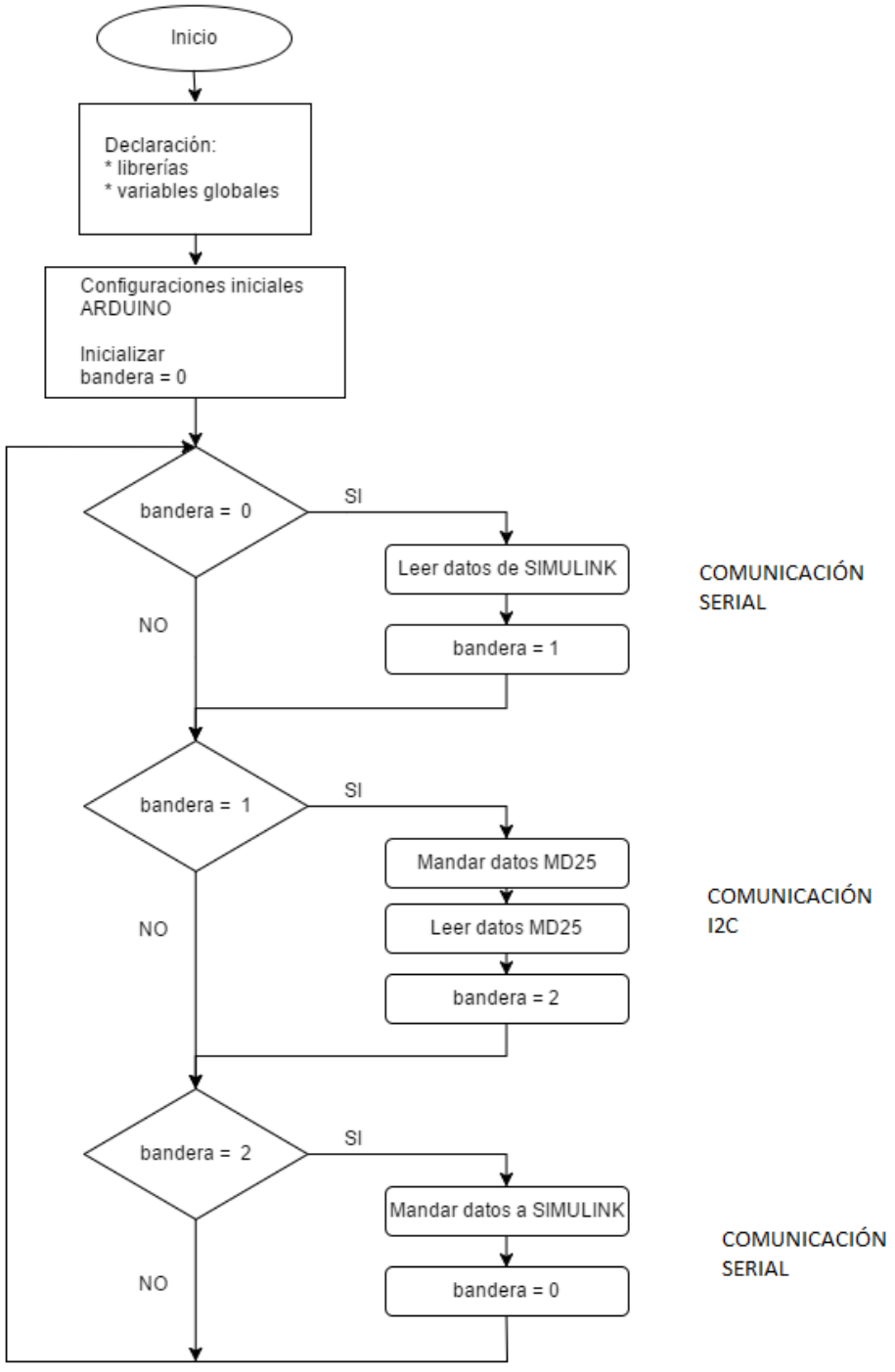


Imagen4.49. Algoritmo para Arduino (Módulo SIMULINK – MD25).

Este algoritmo se programó en el microcontrolador Arduino y se puede consultar en el Apéndice 9 Programas de Arduino, controlador SIMULINK MD25. Como se mencionó, se hará uso de la librería “Wire.h” debido a que se hará uso de la comunicación I2C.

A continuación, se detallan parámetros importantes para realizar de manera adecuada la comunicación que se estableció desde SIMULINK. Se recuerda que SIMULINK envía 3 bytes de información: 1) Dirección I2C de Tarjeta MD25, 2) Velocidad motor A y 3) Velocidad motor B. Para leer dichos Bytes en Arduino se realiza de la siguiente manera (imagen 4.50):

```
#include <Wire.h>

byte Address, SpeedA, SpeedB;

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

void loop()
{
  if(Serial.available()>0)
  {
    Address = Serial.read();
    SpeedA = Serial.read();
    SpeedB = Serial.read();
  }
}
```

Imagen 4.50. Leer los 3 bytes que envía SIMULINK para la MD25en Arduino.

En este caso hay que definir la dirección de los registros en los que se va a escribir o leer de la tarjeta MD25 y asignarles un nombre (imagen 4.51). Las direcciones se pueden consultar en la tabla de control de la Tarjeta.

```
#define SPEED1              0x00
#define SPEED2              0x01
#define ENCODERONE          0x02
#define ENCODERTWO          0x06
#define VOLTREAD            0x0A
#define M1CURRENT           0x0B
#define M2CURRENT           0x0C
#define SOFTWAREREV         0x0D
#define ACCELERATION        0x0E
#define MODE                0x0F
#define CMD                 0x10
#define RESETENCODERS       0x20
#define SPEEDREGULATION     0x31
```

Imagen 4.51. Nombre y dirección de los registros que se usaran de la MD25.

Para mover los motores se creó un método llamado “Move”, el cual requiere tres parámetros: 1) Dirección I^2C de la tarjeta MD25, 2) velocidad del motor A y B. Es necesario declarar el método después del void loop( ) y ahí es en donde se define lo que hará el método con los parámetros recibidos (imagen 4.52).

```
#include <DynamixelSerial1.h>

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

void loop ()
{
  .
  .
   Move(Address,SpeedA,SpeedB);
}

//METODOS//

void Move(byte Address, byte SpeedA, byte SpeedB)
{
  Wire.beginTransmission(byte(Address));
  Wire.write(SPEED1);
  Wire.write(SpeedA);
  Wire.endTransmission();

  Wire.beginTransmission(byte(Address));
  Wire.write(SPEED2);
  Wire.write(SpeedB);
  Wire.endTransmission();
}
```

Imagen 4.52. Declaración del método Move a utilizarse para controlar la MD25.

Posteriormente se hicieron métodos para leer la corriente de cada uno de los motores y el voltaje. Para los encoders primero se debe armar el número int32, en Arduino el equivalente a ese tipo de datos es *long*, por lo cual se debe declarar dos variables tipo *long*, posteriormente se debe separar el numero en 4 bytes, por lo cual se declararon 2 arreglos de dimensión = 4 (imagen 4.53).

```
byte enc1[4], enc2[4];
long poss1, poss2;

void encoder1()
{
  Wire.beginTransmission(byte(Address));
  Wire.write(ENCODERONE);
  Wire.endTransmission();

  Wire.requestFrom(byte(Address), 4);
  while(Wire.available() < 4);

  poss1 = Wire.read();
  poss1 <<= 8;
  poss1 += Wire.read();
  poss1 <<= 8;
  poss1 += Wire.read();
  poss1 <<= 8;
  poss1 +=Wire.read();

  enc1[0] = poss1;
  enc1[1] = poss1>>8;
  enc1[2] = poss1>>16;
  enc1[3] = poss1>>24;
}
```

Imagen 4.53. Método para la lectura y separación de los 32 bits del encoder1 MD25.

Para el método del encoder 2, se debe crear un método similar al encoder 1, sólo se necesita nombrar distinto el método y cambiar el registro *ENCODERONE* por *ENCODERTWO*.

Para los métodos de lectura del voltaje, y corriente de cada uno de los motores, sólo se requiere indicar de qué tarjeta se obtendrá la información, decirle en que registro tiene que hacer la lectura y almacenarlo en una variable como lo muestra la imagen 4.54.

```
byte battery,current1, current2;

void volts()
{
  Wire.beginTransmission(byte(Address));
  Wire.write(VOLTREAD);
  Wire.endTransmission();

  Wire.requestFrom(byte(Address), 1);
  while(Wire.available() < 1);

  battery = Wire.read();
}
```

```
void m1current()
{
  Wire.beginTransmission(byte(Address));
  Wire.write(M1CURRENT);
  Wire.endTransmission();

  Wire.requestFrom(byte(Address), 1);
  while(Wire.available() < 1);

  current1 = Wire.read();
}
```

Imagen 4.54. Método para la lectura de la corriente de los motores 1 y 2 MD25.

Finalmente, se envían los 12 bytes de información de Arduino a SIMULINK, imagen 4.55.

```
Serial.write(Address);
Serial.write(enc1[0]);
Serial.write(enc1[1]);
Serial.write(enc1[2]);
Serial.write(enc1[3]);
Serial.write(enc2[0]);
Serial.write(enc2[1]);
Serial.write(enc2[2]);
Serial.write(enc2[3]);
Serial.write(battery);
Serial.write(current1);
Serial.write(current2);
```

Imagen 4.55. Envió de los 12 bytes de las lecturas de la MD25 a SIMULINK.

De esta manera es como finaliza el diseño del módulo en SIMULINK, que permita enviar y recibir datos de la Tarjeta MD25, cumpliendo con las especificaciones que se establecieron en el Paso 1 del Diseño de Software.

4.3 Fase 3: Evolución

Como establecen la Tabla 4.2, la Fase 3 (Evolución) en el desarrollo de software consiste en darle mantenimiento al software, ya sea para reparar fallos, adaptarlo y mejorarlo o prevenir algún fallo.

El tipo de mantenimiento que se podría emplear sería el *mantenimiento de mejora*, ya que hay características que tienen los Dynamixel y la Tarjeta MD25 que no se incluyeron en este trabajo.

En las tablas 4.18, 4.19 y 4.20 se muestran las tablas de control de los elementos que no se utilizaron en éste proyecto.

Tabla 4.18. Registros que no se utilizaron en este proyecto de la memoria EEPROM Dynamixel.

Area	Address (Hexadecimal)	Name	Description	Access	Valor inicial (Hexadecimal)
EEPROM	0 (0X00)	Model Number(L)	Numero de modelo	R	24 (0X18)
	1 (0X01)	Model Number(H)	Numero de modelo	R	0 (0X00)
	2 (0X02)	Version of Firmware	Información sobre la versión del firmware	R	-
	3 (0X03)	ID	ID del Dynamixel	RW	1 (0X01)
	4 (0X04)	Baud Rate	Velocidad de transmisión de Dynamixel	RW	34 (0X22)
	5 (0X05)	Return Delay Time	Tiempo de retardo	RW	250 (0XFA)
	6 (0X06)	CW Angle Limit(L)	Limite de angulo en sentido horario	RW	0 (0X00)
	7 (0X07)	CW Angle Limit(H)	Limite de angulo en sentido horario	RW	0 (0X00)
	8 (0X08)	CCW Angle Limit(L)	Limite de angulo en sentido antihorario	RW	255 (0XFF)
	9 (0X09)	CCW Angle Limit(H)	Limite de angulo en sentido antihorario	RW	3 (0X03)
	11 (0X0B)	the Highest Limit Temperature	Limite de temperatura interna	RW	80 (0X50)
	12 (0X0C)	the Lowest Limit Voltage	Liminte menor de voltage	RW	60 (0X3C)
	13 (0X0D)	the Highest Limit Voltage	Limite mayor de voltaje	RW	190 (0XBE)
	14 (0X0E)	Max Torque(L)	Max. Torque	RW	255 (0XFF)
	15 (0X0F)	Max Torque(H)	Max. Torque	RW	3 (0X03)
	16 (0X10)	Status Return Level	Estado de devolución de nivel	RW	2 (0X02)
	17 (0X11)	Alarm LED	LED para alarma	RW	36 (0X24)
	18 (0X12)	Alarm Shutdown	Apgado de la alarma	RW	36 (0X24)

Tabla 4.19. Registros que no se utilizaron en este proyecto de la memoria RAM Dynamixel.

Area	Address (Hexadecimal)	Name	Description	Access	Valor inicial (Hexadecimal)
RAM	24 (0X18)	Torque Enable	Torque On/Off	RW	0 (0X00)
	25 (0X19)	LED	LED On/Off	RW	0 (0X00)
	26 (0X1A)	CW Compliance Margin	Margen de cumplimiento sentido horario	RW	1 (0X01)
	27 (0X1B)	CCW Compliance Margin	Margen de cumplimiento sentido antihorario	RW	1 (0X01)
	28 (0X1C)	CW Compliance Slope	Cumplimiento de inclinación sentido horario	RW	32 (0X20)
	29 (0X1D)	CCW Compliance Slope	Cumplimiento de inclinación sentido antihorario	RW	32 (0X20)
	30 (0X1E)	Goal Position(L)	Posicion meta	RW	-
	31 (0X1F)	Goal Position(H)	Posicion meta	RW	-
	32 (0X20)	Moving Speed(L)	Velocidad de movimiento	RW	-
	33 (0X21)	Moving Speed(H)	Velocidad de movimiento	RW	-
	34 (0X22)	Torque Limit(L)	Limite de par (Goal Torque)	RW	ADD14
	35 (0X23)	Torque Limit(H)	Limite de par (Goal Torque)	RW	ADD15
	36 (0X24)	Present Position(L)	Posición actual	R	-
	37 (0X25)	Present Position(H)	Posición actual	R	-
	38 (0X26)	Present Speed(L)	Velocidad actual	R	-
	39 (0X27)	Present Speed(H)	Velocidad actual	R	-
	40 (0X28)	Present Load(L)	Carga actual	R	-
	41 (0X29)	Present Load(H)	Carga actual	R	-
	42 (0X2A)	Present Voltage	Voltaje actual	R	-
	43 (0X2B)	Present Temperature	Temperatura actual	R	-
	44 (0X2C)	Registered	La instrucción se ha registrado	R	0 (0X00)
	46 (0X2E)	Moving	¿Hay akgun movimiento?	R	0 (0X00)
	47 (0X2F)	Lock	Blockeo EEPROM	RW	0 (0X00)
	48 (0X30)	Punch(L)	Punch	RW	32 (0X20)
	49 (0X31)	Punch(H)	Punch	RW	0 (0X00)

Tabla 4.20. Registros que no se utilizaron en este proyecto de la Tarjeta MD25.

Register	Name	Read/Write	Description
0	Speed1	R/W	Motor1 speed (mode 0,1) or speed (mode 2,3)
1	Speed2/Turn	R/W	Motor2 speed (mode 0,1) or turn (mode 2,3)
2	Enc1a	Read only	Encoder 1 position, 1st byte (highest), capture count when read
3	Enc1b	Read only	Encoder 1 position, 2nd byte
4	Enc1c	Read only	Encoder 1 position, 3rd byte
5	Enc1d	Read only	Encoder 1 position, 4th (lowest byte)
6	Enc2a	Read only	Encoder 2 position, 1st byte (highest), capture count when read
7	Enc2b	Read only	Encoder 2 position, 2nd byte
8	Enc2c	Read only	Encoder 2 position, 3rd byte
9	Enc2d	Read only	Encoder 2 position, 4th byte (lowest byte)
10	Battery volts	Read only	The supply battery voltage
11	Motor 1 current	Read only	The current through motor 1
12	Motor 2 current	Read only	The current through motor 2
13	Software Revision	Read only	Software Revision Number
14	Acceleration rate	R/W	Optional Acceleration register
15	Mode	R/W	Mode of operation (see below)
16	Command	Write only	Used for reset of encoder counts and module address changes

Se realizarán pruebas utilizando los módulos de SIMULINK y basándose en los resultados obtenidos se podrá determinar si se necesita darle al software un mantenimiento correctivo o algún otro tipo de mantenimiento. Las pruebas y resultados se detallan en el Capítulo 6.

Capítulo 5

Modelado cinemático

En éste capítulo se desarrolla la cinemática directa e inversa del robot adaptable. Para ello se hace la consideración de que el robot adaptable sólo tiene tres grados de libertad, ya que el análisis que nos interesa es sólo demostrar las variaciones de los parámetros a medir que éste presenta cuando sus eslabones están contraídos (estado normal) y cuando están extendidos (al límite de su final de carrera).

5.1 Cinemática Directa (Denavit Hartenberg)

Para obtener la cinemática directa del robot adaptable, se tomó en cuenta al libro (Barrientos et al., 2007), el cual nos dice que: la forma habitual que se suele utilizar en robótica para la obtención de la cinemática directa en un robot, es la representación de Denavit-Hartenberg (D-H).

Según la representación de D-H, escogiendo adecuadamente los sistemas de coordenadas asociados a cada eslabón, es posible pasar de un eslabón al siguiente mediante cuatro trasformaciones básicas que dependen exclusivamente de las características geométricas del eslabón.

Estas transformaciones básicas consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia de un elemento *i* con los sistemas de un elemento *i-1*. Dichas trasformaciones son las siguientes:

1. La rotación alrededor del eje Z_{i-1} genera un ángulo θi.
2. La traslación a lo largo del eje Z_{i-1} genera una distancia d_i, donde d_i *(0, 0, di)* en un vector.
3. La traslación a lo largo del eje X_i genera una distancia a_i, donde a_i *(0, 0,* a_i*)* es un vector.
4. La rotación a lo largo del eje X_i genera un ángulo α_i.

5.1.1 Algoritmo de Denavit-Hartenberg para la obtención del modelo cinemático directo.

1. Numerar los eslabones comenzando con 1(primer eslabón móvil de la cadena) y acabando con n (último eslabón móvil). Se numerará como eslabón cero a la base fija del robot.
2. Numerar cada articulación comenzando por 1 (La correspondiente al primer grado de libertad) y acabando en n.
3. Localizar el eje de cada articulación. Si ésta es rotativa, el eje será su propio eje de giro. Si es prismática, será el eje a lo largo del cual se produce el desplazamiento.

4. Para $i=0, \ldots, n-1$, situar el eje Z_i sobre el eje de la articulación $i+1$.
5. Situar el origen del sistema de la base $\{S_0\}$ en cualquier punto del eje Z_0. Los ejes X_0 y Y_0, se situarán de modo que formen un sistema dextrógiro con Z_0.
6. Para $i=1, ..., n-1$, se sitúa el sistema de referencia $\{S_i\}$el cual se encuentra unido al eslabón i en la intersección del eje Z_i con la línea normal común a Z_{i-1} y Z_i. Si ambos ejes se cortasen se situaría $\{S_i\}$ en el punto de corte. Si fuesen paralelos $\{S_i\}$ se situaría en la articulación $i+1$.
7. Situar X_i en la línea normal común a Z_{i-1} y Z_i.
8. Situar Y_i de modo que forme un sistema dextrógiro con Xi y Zi.
9. Situar el sistema $\{S_n\}$ en el extremo del robot, de modo que Z_0 coincida con la dirección de Z_{n-1} y X_n normal a Z_{n-1} y Z_n.
10. Obtener θ_i como el ángulo que hay que girar en torno a Z_{i-1} para que X_{i-1} y X_i queden paralelos.
11. Asignar el valor de d_i a la distancia medida a lo lardo de Z_{i-1}, que habría que desplazar al sistema de referencia $\{S_{i-1}\}$ para que X_i y X_{i-1} quedasen alineados.
12. Asignar el valor de a_i a la distancia medida a lo largo de X_i que por los movimientos anteriores ahora coincide con X_{i-1}, realizando un nuevo desplazamiento del eje de referencia $\{S_{i-1}\}$ para que su origen coincida con el eje de referencia de $\{S_i\}$.
13. Asignar el valor de α_i al ángulo que hay que girar en torno a X_i que ahora coinciden con X_{i-1}, para que el nuevo sistema de referencia $\{S_{i-1}\}$ coincida totalmente con el eje de referencia $\{S_i\}$.
14. Obtener las matrices de transformación ${}^{i-1}A_i$.
15. Obtener la matriz de transformación que relacionan al sistema de la base con el del extremo del robot $T= {}^0A_1, {}^1A_2, \ldots, {}^{n-1}A_n$.
16. La matriz T define la orientación (submatriz de rotación) y posición (submatriz de traslación) del extremo referido a la base en función de la n coordenada articulares.

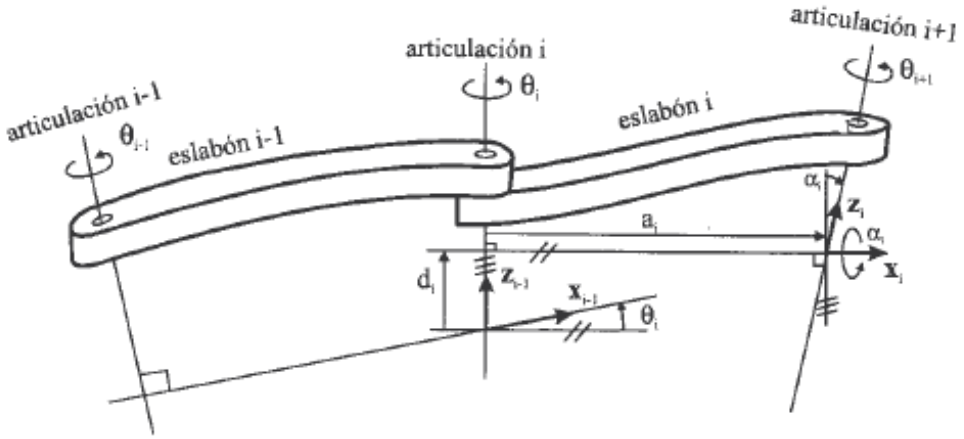


Imagen 5.1. Parámetros D-H para un eslabón giratorio.

5.1.2 Obtención de la cinemática directa del robot adaptable por el método D-H.

En la imagen 5.2 se muestran los pasos del 1 al 4 del método D-H. El paso 1 está de color rojo, el paso 2 está de color amarillo, el paso 3 está de color verde y el paso 4 de color naranja.

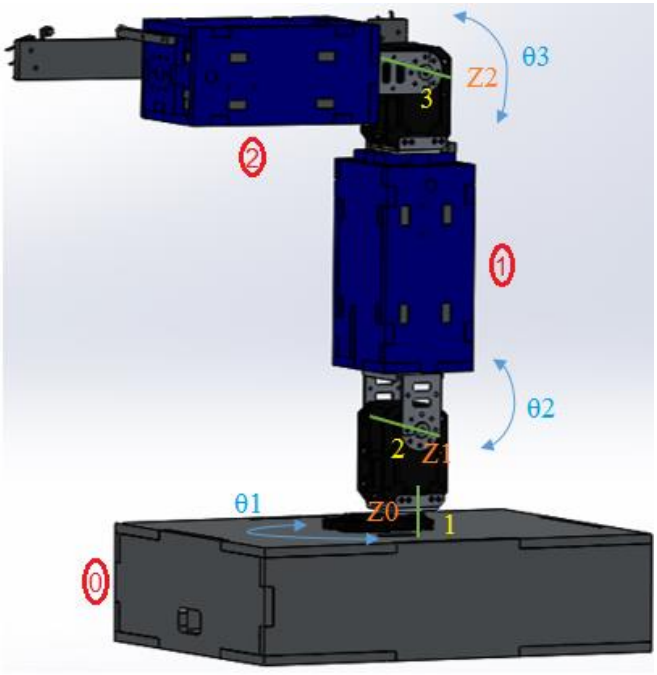


Imagen 5.2. Representación de los pasos 1 a 4 del método de D-H en el robot adaptable.

Los pasos del 5 al 13 del método de D-H se aprecian en la imagen 5.3 y en la tabla 5.1 se obtienen los valores correspondientes a la cinemática directa del robot adaptable.

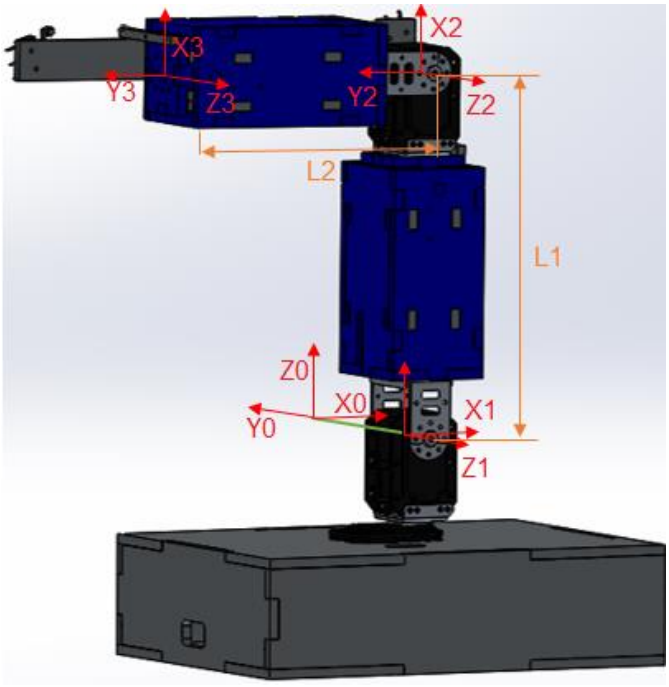


Imagen 5.3. Representación de los pasos del 5 al 13 del método de D-H.

L2=18cm
L3=15cm

Tabla 5.2. Parámetros obtenidos de la cinemática directa del robot adaptable por el método de D-H.

Articulación	θ	d	a	α
1	θ1	0	0	90
2	θ2	0	L1	0
3	θ3	0	L2	0

Luego de obtener los parámetros del método de D-H para el robot adaptable, de la tabla 5.1 se obtienen las matrices *A*, que representan a las reacciones presentes entre los eslabones consecutivos del robot, después de obtener las matrices *A*, es posible obtener las relaciones entre eslabones no consecutivos que están dadas por la matriz *T*.

Como el producto de matrices no es conmutativo, el orden en que deben de realizarse las transformaciones está dada por la siguiente expresión:

$$^{i-1}A_i = T(z, \theta_i)\ T(0,0,d_i)\ T(a_i,0,0)\ T(x, \alpha_i) \qquad [5.1]$$

La expresión anterior, representada matricialmente queda de la siguiente manera:

$$^{i-1}A_i = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} C\theta_i & -C\alpha_i\, S\theta_i & S\alpha_i\, S\theta_i & a_i\, C\theta_i \\ S\theta_i & C\alpha_i\, C\theta_i & -S\alpha_i\, C\theta_i & a_i\, S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [5.2]$$

Sustituyendo los valores de los parámetros obtenidos de la cinemática directa por el método de D-H en la matriz [5.2] es posible obtener las matrices $^0A_1$, $^1A_2$ y $^2A_3$.

$$^0A_1 = \begin{bmatrix} C\theta_1 & 0 & S\theta_1 & 0 \\ S\theta_1 & 0 & -C\theta_1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}; \; ^1A_2 = \begin{bmatrix} C\theta_2 & -S\theta_2 & 0 & L_1C\theta_2 \\ S\theta_2 & C\theta_2 & 0 & L_1S\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$^2A_3 = \begin{bmatrix} C\theta_3 & -S\theta_3 & 0 & L_2C\theta_3 \\ S\theta_3 & C\theta_3 & 0 & L_2S\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Una vez obtenidas las matrices $^0A_1$, $^1A_2$ y $^2A_3$, Se procede a calcular la matriz *T*, que indica la localización del sistema final con respecto al sistema de referencia de la base del robot.

$T = {}^0A_1\, {}^1A_2\, {}^2A_3 =$

$$\begin{bmatrix} C\theta_1\, C(\theta_2+\theta_3) & -C\theta_1\, S(\theta_2+\theta_3) & S\theta_1 & C\theta_1\,(L_2\, C(\theta_2+\theta_3)+L_1\, C\theta_2) \\ S\theta_1\, C(\theta_2+\theta_3) & -S\theta_1\, S(\theta_2+\theta_3) & -C\theta_1 & S\theta_1\,(L_2\, C(\theta_2+\theta_3)+L_1\, C\theta_2) \\ S(\theta_2+\theta_3) & C(\theta_2+\theta_3) & 0 & L_2\, S(\theta_2+\theta_3)+L_1\, S\theta_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

El cálculo de la cinemática directa del robot adaptable, termina con la obtención de la matriz T, ya que ésta es la que se necesita para poder llevar dichas ecuaciones a un software computacional para hacer que el robot adaptable se mueva.

5.2 Cinemática inversa (Método geométrico)

El objetivo principal de calcular la cinemática inversa de un robot, es encontrar los valores que deben tomar las articulaciones de un robot, con la finalidad de que su efector final se posicione y oriente en una determinada localización en el espacio.

Para el cálculo de la cinemática inversa del robot adaptable, se empleó el método geométrico. (Barrientos et al., 2007) Dice que es posible emplear el método geométrico cuando se tienen robots de pocos grados de libertad.

El procedimiento en sí se basa en encontrar suficiente número de relaciones geométricas en las que intervendrán las coordenadas del extremo del robot, sus coordenadas articulares y las dimensiones físicas de sus elementos.

En la imagen 5.4 se muestra la configuración del robot adaptable, así como también las consideraciones a tomar para la obtención de la cinemática inversa. Como se explicó anteriormente, el robot adaptable es de tres grados de libertad y por lo tanto el método geométrico se puede emplear para encontrar la solución de la cinemática inversa.

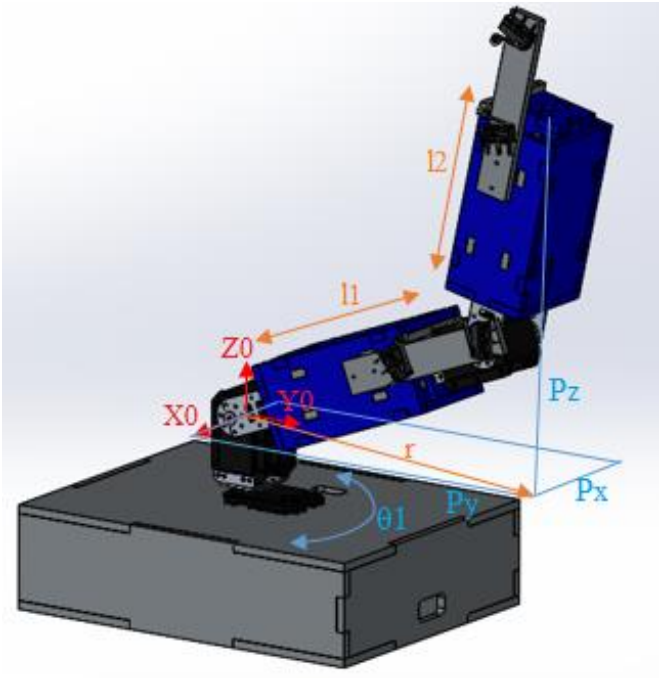


Imagen 5.4. Consideraciones para la obtención de la cinemática inversa.

Para comenzar con el método, se definirán las coordenadas (Px, Py, Pz) referidas al sistema {S0} como se muestra en la imagen 5.4. Es posible obtener el valor de θ_1 directamente, ya

que éste gira sobre el eje Z y por lo tanto se puede calcular con ayuda de la función trigonométrica de la tangente, mostrada en la imagen 5.5.

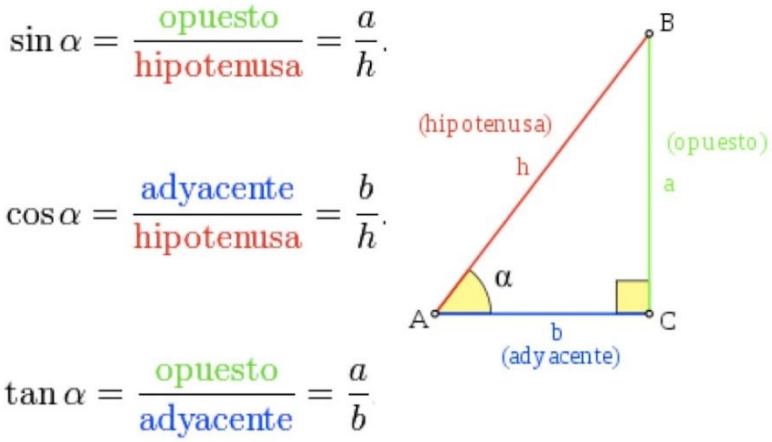


Imagen 5.5. funciones trigonométricas respecto de un triángulo rectángulo.

$$\tan\theta_1 = \left(\frac{P_y}{P_x}\right)$$

$$\theta_1 = arctg\left(\frac{P_y}{P_x}\right) \qquad [5.3]$$

La ecuación [5.3] da el valor de la cinemática inversa para θ_1.

Ahora considerando únicamente a los elementos extremos del robot adaptable L_1 y L_2, es posible obtener a θ_3 y θ_2. Como se ve en la imagen 5.6, se obtienen dos soluciones para θ_3, dependiendo de si se toma el valor positivo o negativo del resultado de la raíz. Dicho signo corresponde a las configuraciones de codo arriba y codo abajo.

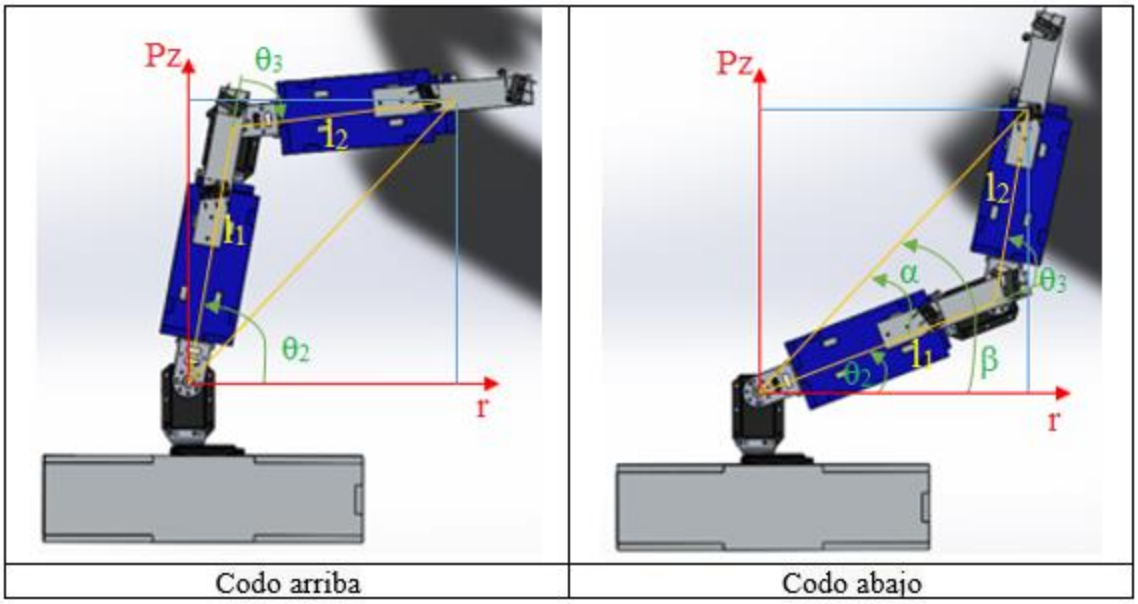
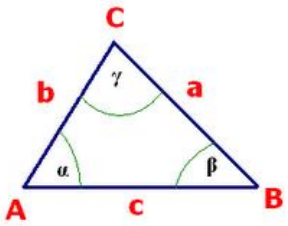


Imagen 5.6. Representación del robot adaptable en el plano para la obtención de θ_2 y θ_3.

Tomando como base la imagen 5.6, es posible hacer un análisis en el plano, y así obtener las ecuaciones que resuelvan la cinemática inversa para θ_2 y θ_3 . Lo anterior simplifica la obtención de las ecuaciones ya que sólo se tienen articulaciones rotacionales sobre un sólo plano.

De la imagen 5.4 se sabe que: $r^2 = P_x^2 + P_y^2$ [5.4] y utilizando la ley de cosenos (imagen 5.7) para el triángulo que se forma entre los eslabones l_1 y l_2 que se muestra en la imagen 5.5 (de cualquier configuración, codo abajo o codo arriba):



$$a^2 = b^2 + c^2 - 2bc \cdot \cos\alpha$$

$$b^2 = a^2 + c^2 - 2ac \cdot \cos\beta$$

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

Imagen 5.7. Ley de cosenos.

De la ley de cosenos mostrada en la imagen 5.7, se remplazan los valores, por las variables empleadas en el robot adaptable, pero considerando el signo de la expresión $-2l_1 l_2 \cos\theta_3$ como positivo, puesto que, si se considera como negativo, se estaría calculando el ángulo comprendido entre l_1 y l_2 y no el mostrado en la imagen 5.6 en configuración codo abajo como θ_3. Como resultado de lo anterior, la ecuación queda de la siguiente manera:

$r^2 + P_z^2 = l_1^2 + l_2^2 + 2l_1 l_2 \cos\theta_3$ [5.5].

Se sustituye la ecuación [5.4] en [5.5] y se obtiene: $\cos\theta_3 = \frac{P_x^2 + P_y^2 + P_z^2 - l_1^2 - l_2^2}{2l_1 l_2}$ [5.6]

De la ecuación [5.6] es posible obtener el valor de θ_3, en función del vector de posición P, sin embargo, es más conveniente tener la expresión en términos de arcotangente en lugar de una función arcoseno, ya que se obtienen grandes ventajas al momento de meter las ecuaciones en un software de cálculo.

Con base en la identidad trigonométrica [5.7] es posible expresar la ecuación 3 en términos de arcotangente.

$$\text{sen}^2\,\theta + \cos^2\theta = 1$$

$$\text{sen}\,\theta = \sqrt{1 - \cos^2\theta} \qquad [5.7]$$

Sustituyendo las ecuaciones [5.6] y [5.7] en la ecuación [5.8], se obtiene:

$$\tan\theta_3 = \frac{\text{sen}\,\theta_3}{\cos\theta_3} \qquad [5.8]$$

$$\tan\theta_3 = \frac{\text{sen}\,\theta_3}{\cos\theta_3} = \frac{\sqrt{1 - \cos^2\theta_3}}{\cos\theta_3}$$

$$\theta_3 = arctg\left(\frac{\sqrt{1-\cos^2\theta_3}}{\cos\theta_3}\right) \qquad [5.9]$$

Con: $\cos\theta_3 = \frac{P_x^2 + P_y^2 + P_z^2 - l_1^2 - l_2^2}{2l_1 l_2}$

Como se observa en la imagen 5.6 existen dos posibles soluciones para θ_3, según se tome el signo positivo o negativo que se obtenga como resultado de la solución de la raíz.

De la imagen 5.6 en configuración codo abajo, es posible obtener la ecuación que sirve para encontrar el valor de θ_2, de la siguiente manera:

$$\theta_2 = \beta - \propto \qquad [6.0]$$

Para la obtención de los valores de $\propto y\ \beta$, es necesario apoyarse en la imagen 5.6, de donde se obtiene:

$$\tan\beta = \left(\frac{P_z}{r}\right)$$

$$\beta = arctg\left(\frac{P_z}{r}\right) = arctg\left(\frac{P_z}{\sqrt{P_x^2 + P_y^2}}\right) \qquad [6.1]$$

Con la ecuación [6.1] es posible obtener el valor de β. Para obtener el valor de $\propto$, es necesario apoyarse en la imagen 5.6 y hacer la consideración de que el único ángulo que hace que se modifique el valor de $\propto$, es el ángulo θ_3. Por lo anterior, el análisis se obtiene como:

$$\tan \propto = \left(\frac{l_2 \operatorname{sen}\theta_3}{l_1 + l_2\cos\theta_3}\right)$$

$$\propto = arctg\left(\frac{l_2 \operatorname{sen}\theta_3}{l_1 + l_2\cos\theta_3}\right) \qquad [6.2]$$

Finalmente sustituyendo las ecuaciones [6.1] y [6.2] en $\theta_2 = \beta - \propto$, se obtiene:

$$\theta_2 = arctg\left(\frac{P_z}{\sqrt{P_x^2 + P_y^2}}\right) - arctg\left(\frac{l_2 \operatorname{sen}\theta_3}{l_1 + l_2\cos\theta_3}\right) \qquad [6.3]$$

Con las ecuaciones [5.3], [5.9] y [6.3], se resuelve la cinemática inversa para el robot adaptable, pero para meter dichas ecuaciones en un software computacional que haga que el robot se mueva correctamente, es necesario expresar las ecuaciones en términos de Atan2.

En una variedad de lenguajes de programación, el Atan2 es la función arco tangente pero que contiene dos argumentos, con el propósito de obtener mayor información sobre los signos que arrojan los resultados, para así proporcionar el cuadrante correcto al ángulo calculado. Atan2(y,x) es el ángulo en radianes calculado entre el eje x positivo y un punto dado (x, y), el ángulo es positivo para ángulos en sentido anti horario (semiplano superior, y>0), y negativo para ángulos en sentido horario (semiplano inferior y<0).

5.3 Programación de la cinemática directa e inversa

La programación de la cinemática directa e inversa se realizó en MATLAB. En dicho programa se utilizaron Script para cada ángulo obtenido, quedando estos de la siguiente manera:

Antes de comenzar, es importante señalar que para la programación se cambiaron algunos símbolos por su nombre, tal es el caso de θ_1 que se cambió por theta 1, θ_2 por theta 2 y θ_3 por theta 3, con el fin de simplificar dicha programación. Dicho lo anterior, para theta 1 sólo se requieren dos entradas (x,y) y al final se obtiene una salida como lo establece la ecuación [5.3].

$$\theta_1 = arctg\left(\frac{P_y}{P_x}\right) \qquad [5.3]$$

Sin embargo, para la programación en MATLAB se requieren dos salidas, la primera es el resultado que sale de las ecuaciones obtenidas y el segundo valor es de un ajuste que se requiere, debido a que los ejes de referencia del modelo matemático no corresponden a los ejes de referencia de los servomotores como se puede apreciar en la Imagen 5.10.

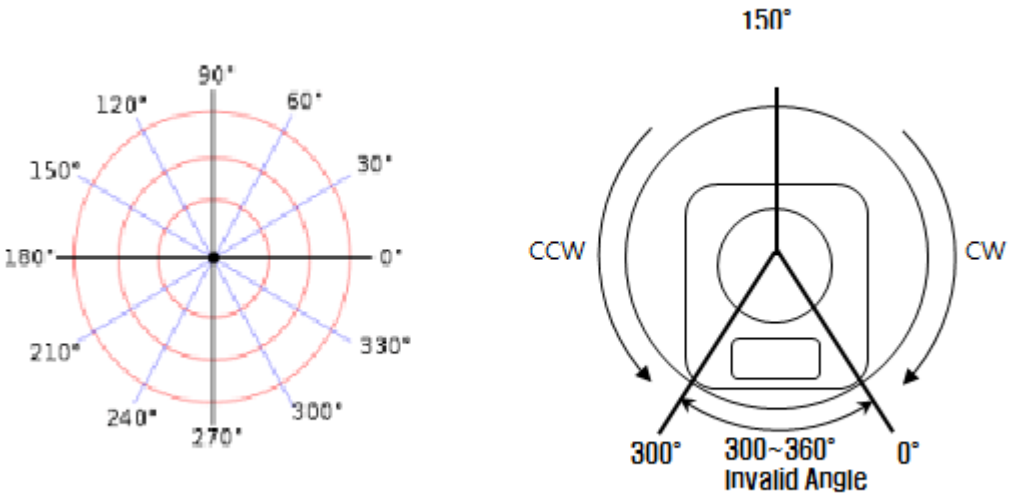


Imagen 5.10 Sistema de referencia teórico y sistema de referencia de los Dynamixel.

Por lo cual el script “theta1.m” queda como se muestra en la Imagen 5.11 y para usarse en el *Interpreted MATLAB Function* se configuran dos entradas y dos salidas como se muestra en la Imagen 5.12.

theta1.m

```
1   function sal = theta1(x,y)
2
3 -   t1 = atand(y/x);
4 -   salida = [t1;60+t1];
5
6 -   sal = salida;
```

Imagen 5.11 Script theta 1.

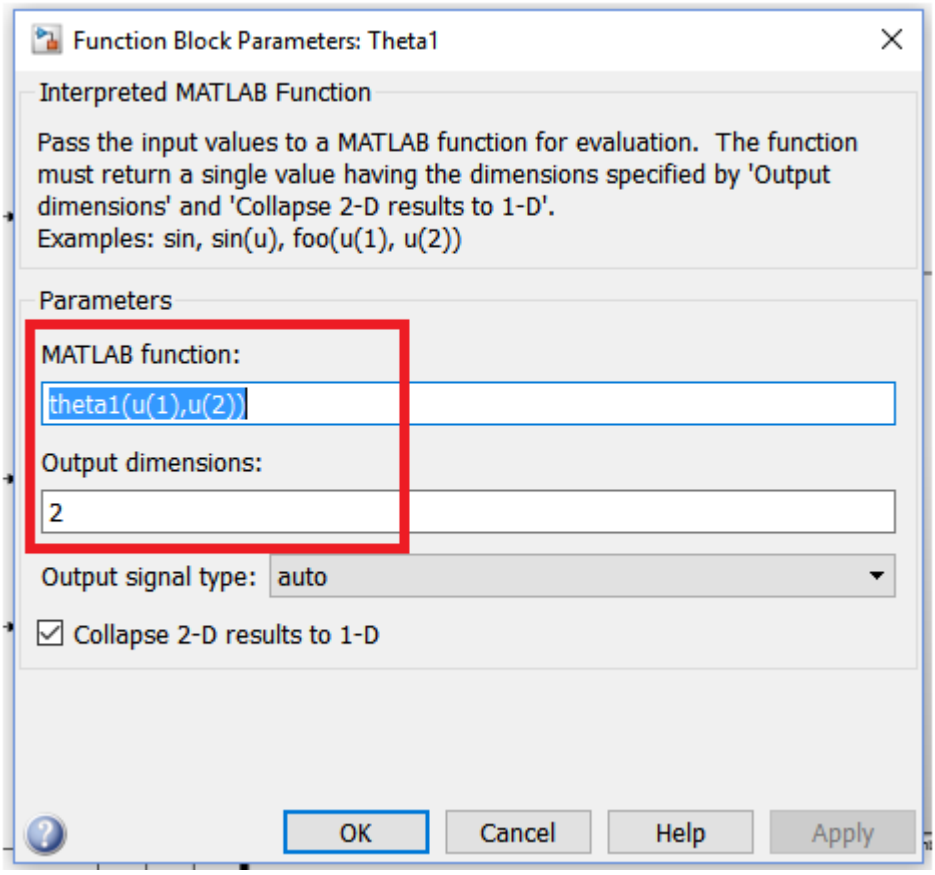


Imagen 5.12. Configuración del script theta1 en el *Interpreted MATLAB Function*.

Para theta 3 se requieren 5 entradas (Px, Py, Pz, l_1 y l_2), como lo establece la ecuación [5.9] y al final se obtienen dos salidas como en el caso anterior, quedando el script como lo muestran las Imagenes 5.13 y 5.14.

$$\theta_3 = arctg\left(\frac{\sqrt{1-\cos^2\theta_3}}{\cos\theta_3}\right) \qquad [5.9]$$

$$\text{Con: } \cos\theta_3 = \frac{P_x^2+P_y^2+P_z^2-l_1^2-l_2^2}{2l_1l_2}$$

```
theta3.m
1  function sal = theta3(x,y,z,l2,l3)
2
3  cq3 = (x^2 + y^2 + z^2 - l2^2 - l3^2)/(2*l2*l3);
4
5  t3 = atan2d(-sqrt(1-(cq3)^2),cq3);
6  salida = [t3;150+t3];
7
8  sal = salida;
```

Imagen 5.13. Script theta3.

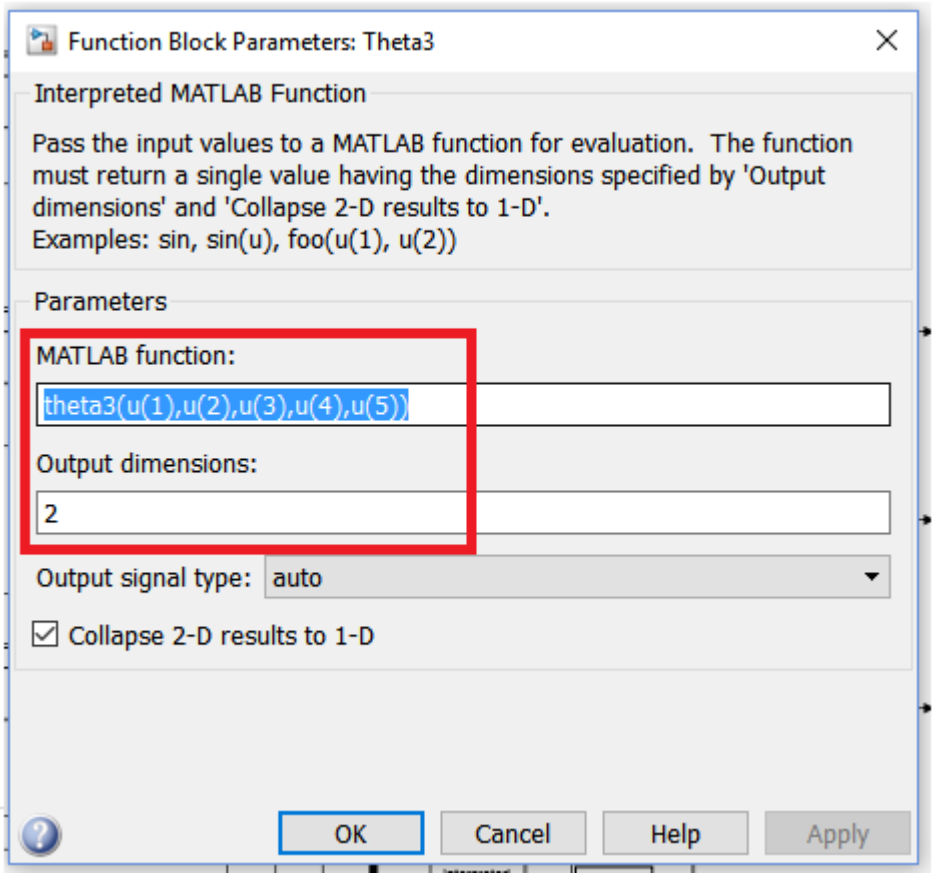


Imagen 5.14. Configuración del script theta3 en el *Interpreted MATLAB Function*.

Para theta 2 se requieren 5 entradas (Px, Py, Pz, l_1 y l_2), como lo establece la ecuación [6.3] y al final se obtienen dos salidas como en el caso anterior, quedando el script como lo muestran las imágenes 5.15 y 5.16.

$$\theta_2 = arctg\left(\frac{P_z}{\sqrt{P_x^2+P_y^2}}\right) - arctg\left(\frac{l_2 \operatorname{sen}\theta_3}{l_1+l_2\cos\theta_3}\right) \qquad [6.3]$$

theta2.m

```
1   function sal = theta2(x,y,z,l2,l3)
2
3   cq3 = (x^2 + y^2 + z^2 - l2^2 - l3^2)/(2*l2*l3);
4   sq3 = -sqrt(1-cq3^2);
5
6
7   t2 = atan2d(z,sqrt(x^2+y^2))- atan2d((l3*sq3),(l2+l3*cq3));
8
9   salida = [t2;60+t2];
10
11  sal = salida;
```

Imagen 5.15. Script theta 2.

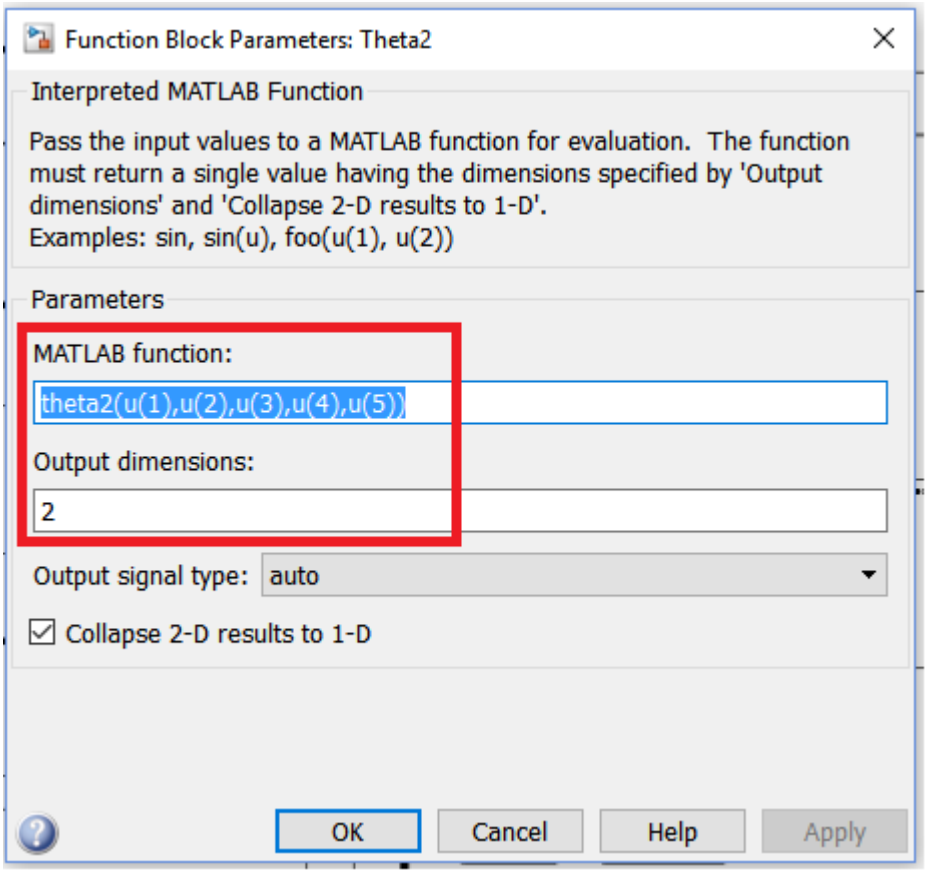


Imagen 5.16. Configuración del script theta2 en el *Interpreted MATLAB Function*.

NOTA: Se conservaron ambos valores de salida (valor teórico y valor con ajuste). El primer valor sirve para poder hacer una simulación en MATLAB del robot moviéndose en tiempo real y el segundo valor es el que se le envía al robot mediante el uso de los módulos de SIMULINK-Dynamixel.

Capítulo 6

Pruebas y resultados

Un Banco de pruebas es una instalación que sirve para medir variables y características de funcionamiento de prototipos. Los bancos de pruebas brindan una forma de comprobación rigurosa, transparente y repetible de teorías científicas(Constructora, 2016).

En éste capítulo se realizan dos pruebas que permiten usar al robot adaptable como un banco de pruebas, utilizando los módulos SIMULINK-Dynamixel y SIMULINK-MD25 que se diseñaron en el Capítulo 4.

Objetivo general de las pruebas:

El robot adaptable debe de ser capaz de seguir una trayectoria preestablecida de manera autónoma, y además con ayuda del módulo SIMULINK-Dynamixel obtener las lecturas medidas directamente de los servomotores, tales como: posición actual, velocidad actual, porcentaje de carga, sentido de carga, voltaje y temperatura. Además de utilizar el módulo SIMULINK-MD25 para verificar el correcto funcionamiento de estos módulos al hacer lecturas de voltaje y corriente de cada micro- motorreductor Pololu o incluso cualquier otro motor DC con encoder que se desee utilizar.

Objetivos particulares:

1. Diseñar una prueba general para cada uno de los módulos de manera que puedan medir variables y determinar el correcto funcionamiento.
2. Cada prueba debe ser repetible, con la finalidad de realizar distintas pruebas que permitan obtener datos que resulten de interés en el área de los robots adaptables.
3. Las pruebas deben estar diseñadas de manera modular, por si se requiere modificar, ampliar o disminuir el alcance de la prueba, se pueda agregar un nuevo módulo, eliminar o modificar los módulos existentes y esto no resulte difícil de hacer.

6.1 Descripción de la prueba 1 (SIMULINK – Dynamixel):

La prueba consiste en hacer que el robot adaptable siga la trayectoria de una elipse trazada en el espacio. Dicha prueba se realiza dentro de un campo de trabajo apto para el robot adaptable, es decir, donde sea capaz de moverse con facilidad. Sin embargo, la elipse se puede trasladar y rotar conforme el usuario necesite realizar las pruebas para obtener datos y características del funcionamiento del robot adaptable.

Para la realización de esta prueba, fue necesario trabajar con 6 módulos realizados en SIMULINK. El primer módulo genera una curva en el plano; el segundo módulo rota y traslada en el espacio dicha curva; el tercer módulo contiene la cinemática inversa del robot, el cuarto módulo es un selector de datos (Ingresan 6 datos y van saliendo en pares); el quinto módulo es el encargado de enviar la posición de cada uno de los servos y mostrar las lecturas obtenidas de cada uno de los servomotores y finalmente el sexto módulo muestra la simulación del movimiento del robot adaptable en tiempo real. En la Imagen 6.1 se muestran los módulos antes mencionados.

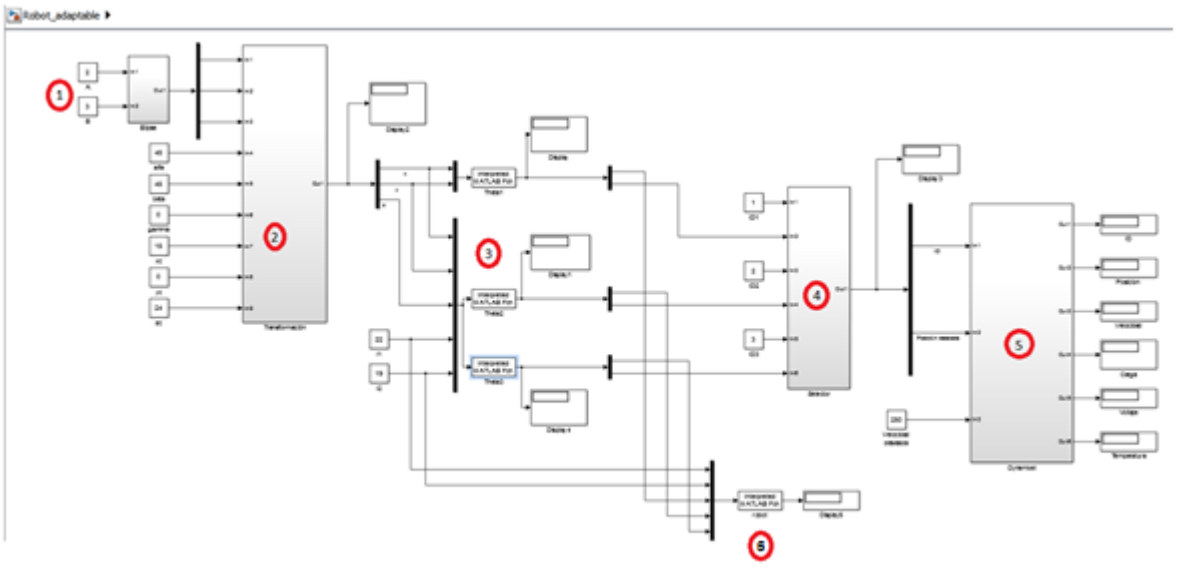


Imagen 6.1. Configuración en SIMULINK para realizar la Prueba 1.

6.2 Funcionamiento de los módulos para realizar la prueba 1

Módulo 1 (Curva en el plano)

En éste primer módulo genera una curva en el plano, en este caso se genera una elipse. Para lo cual fue necesario introducir la ecuación parametrica de una elipse en SIMULINK, como se muestra acontinuacion:

$$x = A * sen(\theta)$$

$$y = B * cos(\theta)$$

$$z = 0$$

En la imagen 6.2 se observa el módulo 1 que se diseñó en SIMULINK, como se puede apreciar el módulo tiene dos parametros de entrada, los valores de A y B de la elipse, con dichos valores se puede modificar la amplitud sobre su eje mayor y eje menor. A la salida del módulo se obtienen tres valores, que corresponden a los puntos por donde pasa la elipse en los ejes x,y,z.

NOTA: La salida de los puntos en el eje z siempre es 0 debido a que es una curva en el plano, sin embargo, se consideró tener una salida en el eje Z por si a futuro se requiere una curva en el espacio en lugar de una curva en el plano.

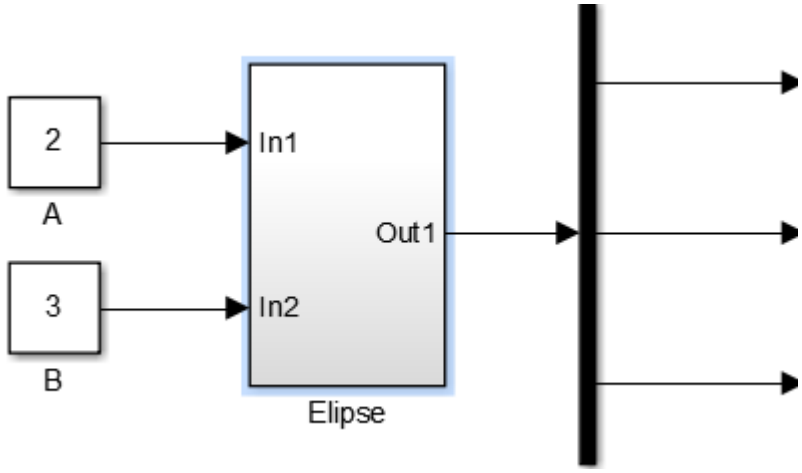


Imagen 6.2. Módulo 1 (Curva en el plano).

Módulo 2 (Curva en el espacio)

Los valores *(x,y,z)* que salen del módulo 1 ingresan al módulo 2, donde se encuentra la matriz de rotación y traslación en el espacio, matriz [6.1]. Para que el módulo 2 funcione correctamente, es necesario ingresar 9 entradas *(x, y, z, alfa, beta, gamma, Xc, Yc, Zc).*

- *(x,y,z):* son los valores que salen del módulo 1 y que corresponden a los puntos por donde pasa la elipse.
- *Alfa*: representa al ángulo de rotación sobre el eje *X*, del cual se quiere que esté girada la elipse.
- *Beta*: representa al ángulo de rotación sobre el eje *Y*, del cual se quiere que esté girada la elipse.
- *Gamma*: representa al ángulo de rotación sobre el eje *Z*, del cual se quiere que esté girada la elipse.
- *(Xc, Yc, Zc):* son los valores del centro de la elipse en el espacio, que corresponden a la traslacion del centro de ésta.

$$\begin{bmatrix} C[\beta]C[\gamma] & -C[\beta]S[\gamma] & S[\beta] & x_c \\ C[\gamma]S[\alpha]S[\beta] + C[\alpha]S[\gamma] & C[\alpha]C[\gamma] - S[\alpha]S[\beta]S[\gamma] & -C[\beta]S[\alpha] & y_c \\ -C[\alpha]C[\gamma]S[\beta] + S[\alpha]S[\gamma] & C[\gamma]S[\alpha] + C[\alpha]S[\beta]S[\gamma] & C[\alpha]C[\beta] & z_c \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [6.1]$$

A la salida del módulo 2 se obtiene (Px,Py,Pz), que son los puntos por donde pasa la elipse en el espacio, despúes de haber sido rotada y trasladada, en la imagen 6.3 se pueden ver como están conecatdos los módulos 1 y 2.

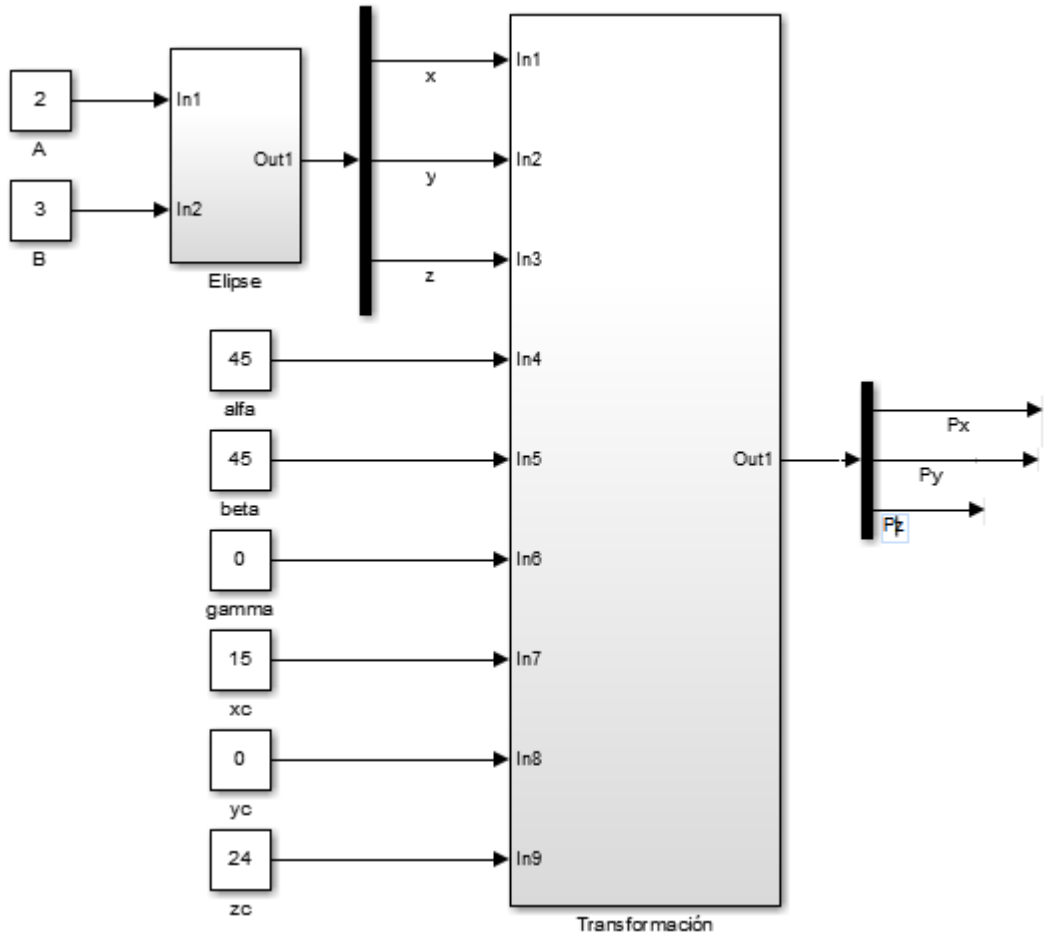


Imagen 6.3. Módulo 2 (Curva en el espacio).

Módulo 3 (Cinemática Inversa)

Una vez obtenidos los valores de *(Px,Py,Pz)*, que son los puntos por donde pasa la elipse en el espacio, se procede a obtener la cinemática inversa del robot adaptable, para ello se necesitan los valores de *Px,Py,Pz, l_1 y l_2*, donde l_1 y l_2 corresponden a las medidas del eslabón 1 y 2 respectivamente.

Como se vio en el capítulo 5 (cinemática inversa), para obtener a theta 1 sólo se necesitan los valores de *Px y Py*, y para los valores de theta 2 y theta 3 es necesario tener los valores de *Px,Py,Pz*, l_1 *y* l_2 (imagen 6.4).

Como se vio en el capítulo 5 (programación de la cinemática directa e inversa), se crearon 3 script para determinar los valores de theta1, theta2 y theta 3, donde cada uno tenía dos entradas, seis entradas y seis entradas respectivamente y los 3 script tenían 2 salidas.

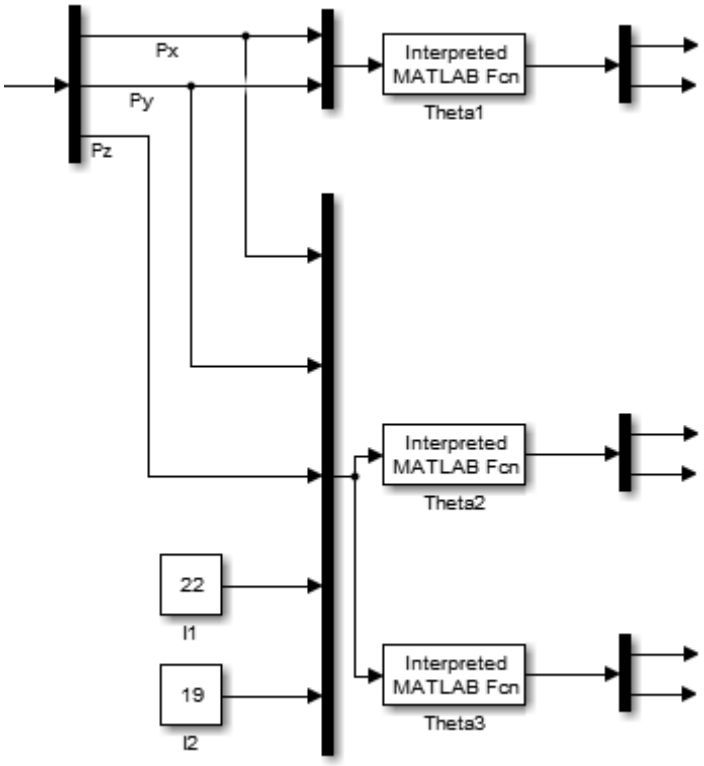


Imagen 6.4. Módulo 4 (Cinemática Inversa).

Módulo 4 (Selector)

Éste módulo tiene 6 entradas y va sacando un par de datos cada 0.1 [s]. La función principal de éste módulo es seleccionar a qué servomotor le enviará la información necesaria para posicionarlo en el valor que le asigne la cinemática inversa.

Los datos de entrada son el ID (1, 2, 3) los cuales son valores constantes y la posición a la que se moverán los servomotores Dynamixel (Pos ID1, Pos ID2, Pos ID3); a la salida se obtiene el ID del Servo a mover y la posición que se calculó en la cinemática inversa (imagen 6.5). Cada 0.5 s el sector cambia de ID1 Pos ID1 a ID2 Pos ID2 después a ID3 Pos ID3, para

volver a comenzar con el barrido de lecturas con ID1 Pos ID1 y así sucesivamente; con la finalidad de poder mover todos los servomotores casi al mismo tiempo y a su vez ir obteniendo los valores de las lecturas de interés de cada uno, en cada posicionamiento de estos al seguir la trayectoria de la elipse.

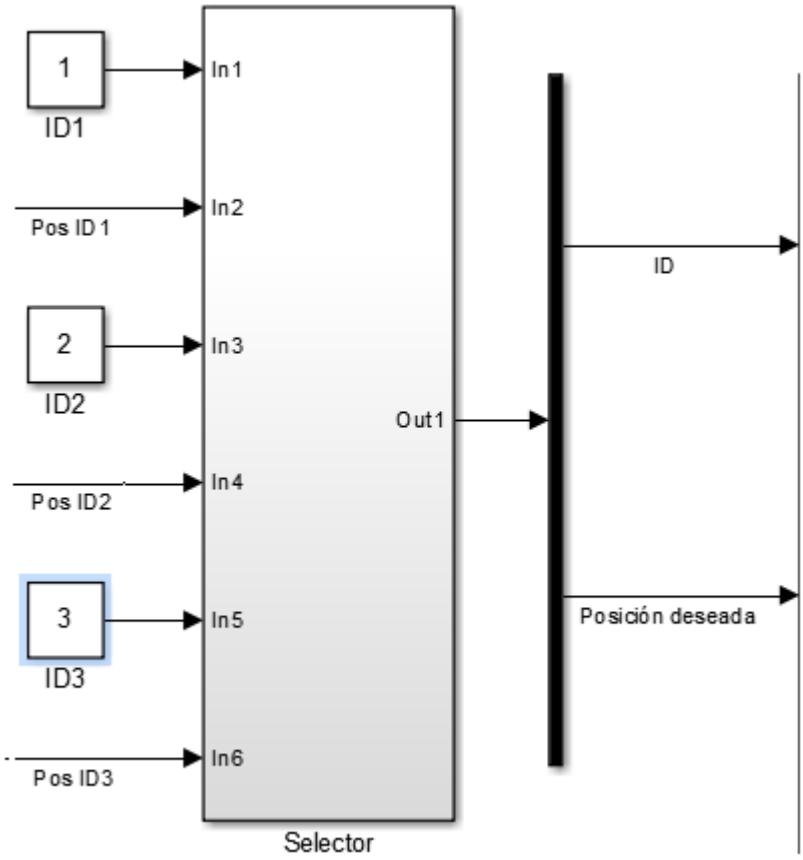


Imagen 6.5. Módulo 5 (Selector).

Módulo 5 (SIMULINK-Dynamixel)

Aquí se emplea el módulo de comunicación que se creó en el Capítulo 4 (SIMULINK-Dynamixel), el cual a la entrada le llegan los datos de a que servomotor va a mover y la velocidad con la que éste se moverá, a la salida se obtienen 6 valores: ID, posición, velocidad, carga, voltaje y temperatura, dicho módulo se puede observar en la imagen 6.6.

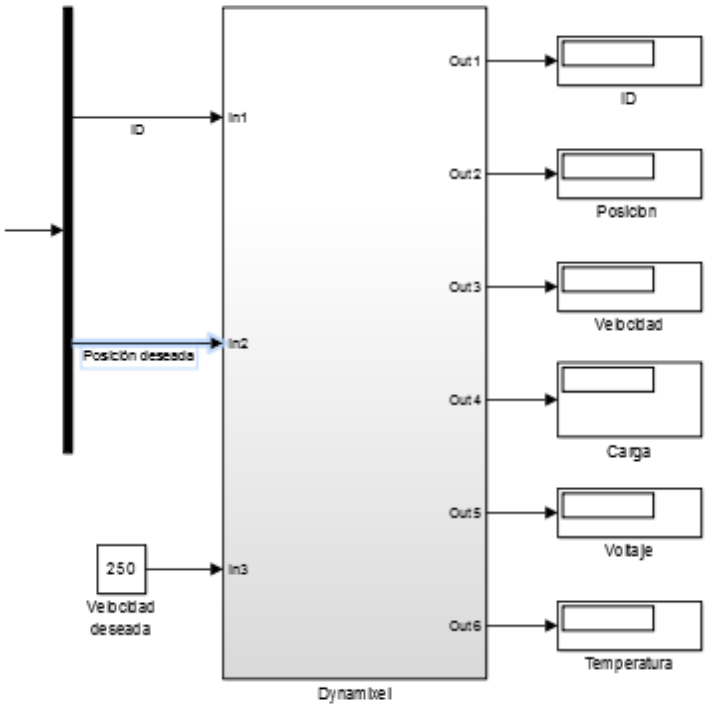


Imagen 6.6. Módulo 5 (SIMULINK-Dynamixel).

Módulo 6 (Simulación robot)

Para la simulación del robot se utilizaron las longitudes de los eslabones l_1 y l_2, así como también los ángulos theta1, theta2 y theta3 que se obtuvieron de la cinemática inversa, de igual manera se tomó en cuenta a la cinemática directa obtenida del robot adaptable. Se diseñó un script “robot.m” (imagen 6.7) con la finalidad de tener una simulación en tiempo real de las pruebas para el robot adaptable y así mismo ver cómo se mueve éste al ejecutar la trayectoria deseada (imagen 6.8).

robot.m

```
function sal = robot(l1,l2,th1,th2,th3)

th1=degtorad(th1);
th2=degtorad(th2);
th3=degtorad(th3);

s0=eye(4);
s1=s0*rotz(th1);
s2=s1*rotx(pi/2)*rotz(th2);
s3=s2*transl(l1,0,0)*rotz(th3);
s4=s3*transl(l2,0,0);

x=[s0(1,4) s1(1,4) s2(1,4) s3(1,4) s4(1,4)];
y=[s0(2,4) s1(2,4) s2(2,4) s3(2,4) s4(2,4)];
z=[s0(3,4) s1(3,4) s2(3,4) s3(3,4) s4(3,4)];

plot3(x,y,z)
title('Trayectoria del robot')
xlabel('Eje X')
ylabel('Eje Y')
zlabel('Eje Z')
grid on
axis([-35 35 -35 35 0 35])

sal = 0;
```

Imagen 6.7. Script robot.

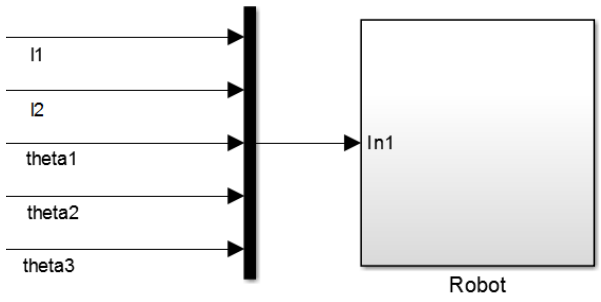


Imagen 6.8 Módulo 6. (Simulación robot).

6.3 Ejecución de la prueba 1

Los 6 módulos anteriormente descritos, trabajando en conjunto, permiten realizar la prueba y obtener los datos de interés de los Dynamixel, mientras el robot adaptable ejecuta la trayectoria.

Los parámetros que se utilizaron son los que se aprecian en las tablas 6.1, 6.2 y 6.3.

Tabla 6.1. Valores de A y B para la elipse.

Elipse	
A	2
B	3

Tabla 6.2. Parámetros para rotar y trasladar la elipse.

Transformación	
alfa	90
beta	90
gamma	0
x	13
y	0
z	22

Tabla 6.3. Longitudes de los eslabones adaptables para la prueba 1.

Longitud Eslabones	
l1	18
l2	15

Al correr la prueba con los datos de las tablas 6.1, 6.2 y 6.3, el robot adaptable se movió siguiendo la trayectoria de la elipse en el espacio, y de manera simultánea se iba generando la simulación (imagen 6.9). Al final, cuando el robot terminó de seguir la trayectoria, se obtuvieron los valores de interés para cada uno de los servomotores (imagen 6.10).

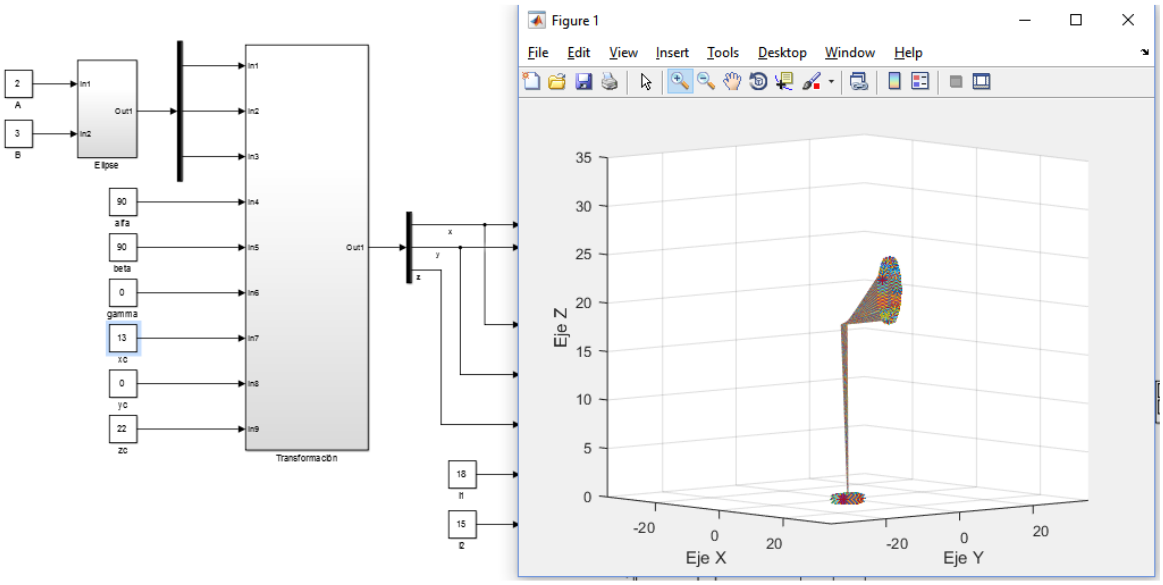


Imagen 6.9 Simulación del robot al terminar la Prueba.

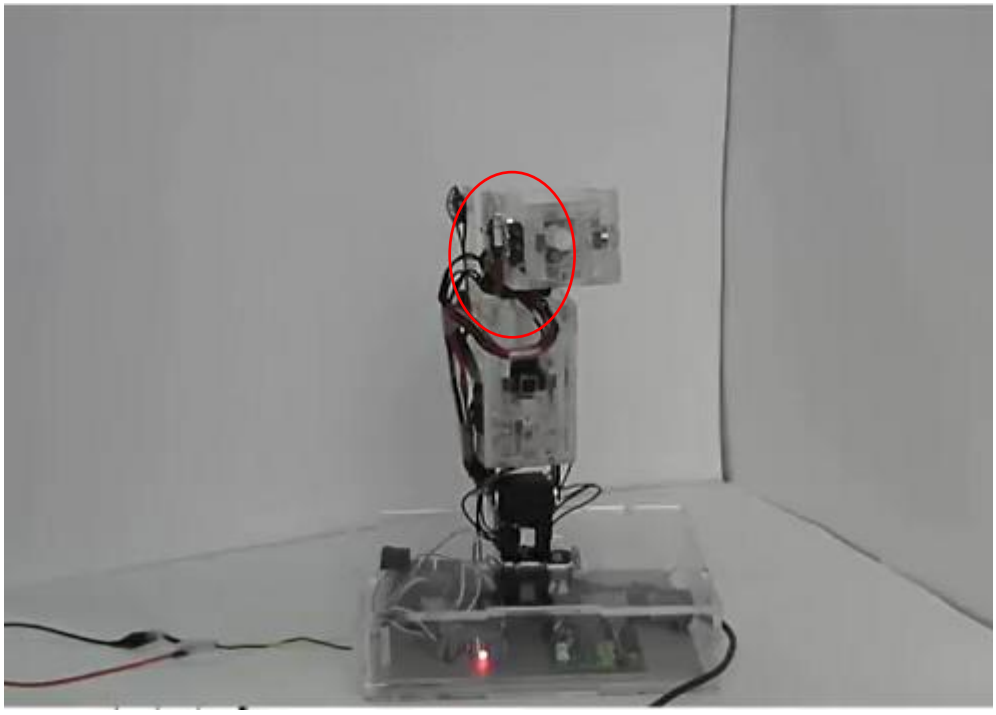


Imagen 6.10 Robot adaptable siguiendo la trayectoria propuesta.

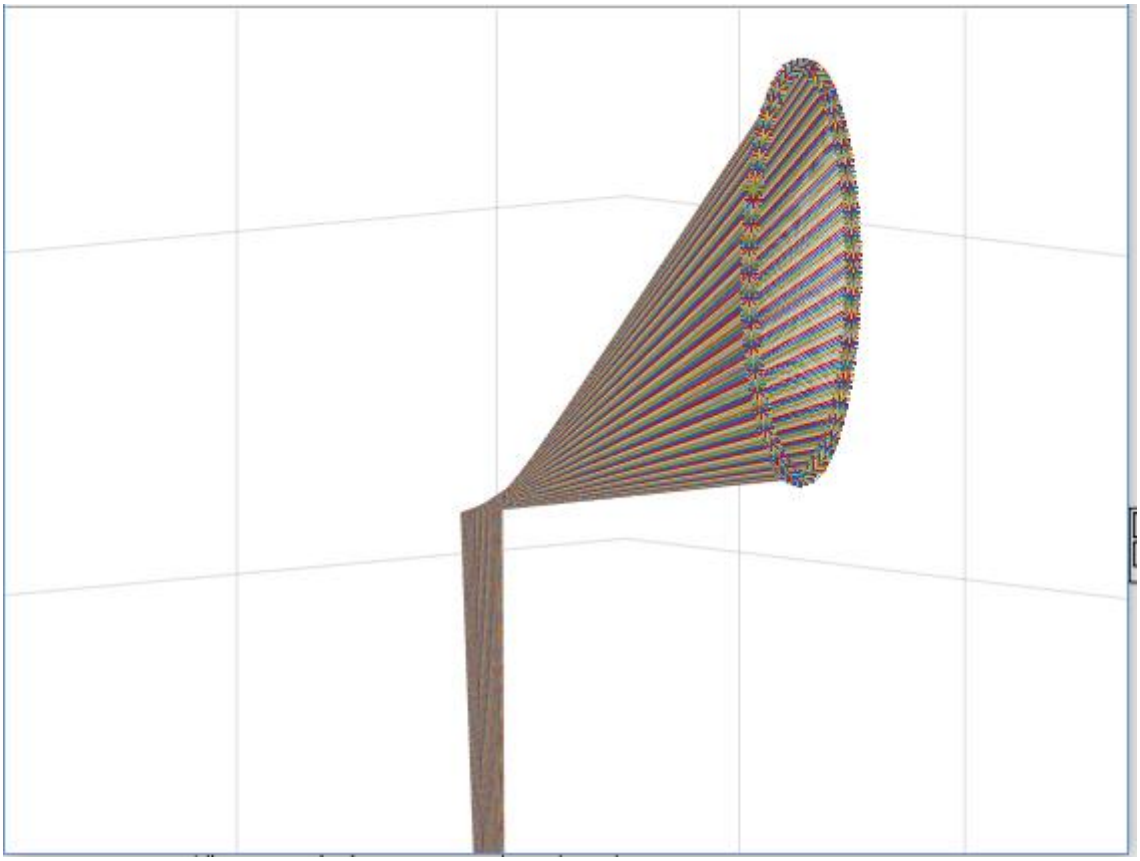


Imagen 6.11 Zoom de la Simulación del robot al terminar la Prueba

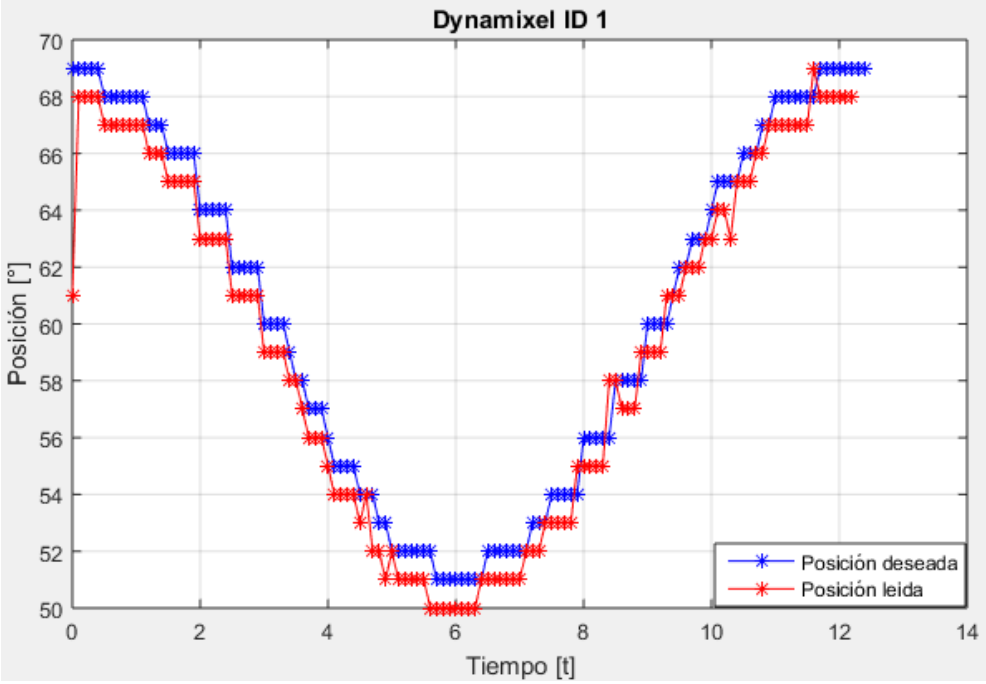
Una vez terminada la prueba, se guardan los datos enviados y leídos de los Dynamixel en el Workspace de MATLAB, esos datos se almacenaron en una hoja de cálculo para posteriormente procesarlos con un script en MATLAB, llamado Graficas_Dynamixel.m, el cual genera 18 graficas, 6 por cada servomotor:

1. Posición deseada y posición leída
2. Velocidad
3. Porcentaje de carga
4. Sentido de giro
5. Voltaje
6. Temperatura

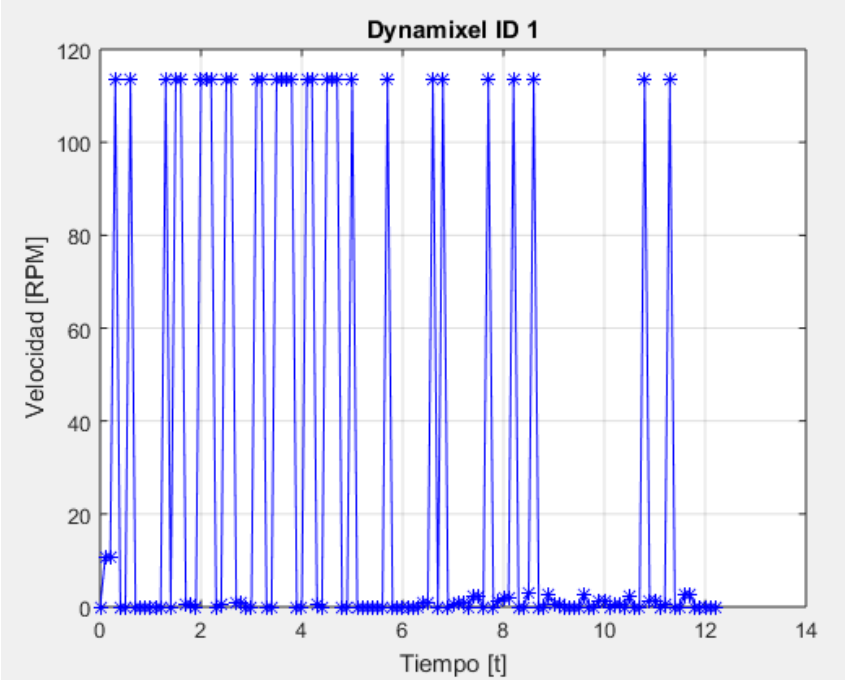
El script se encuentra en el Apéndice 10 (Script para generar gráficas).

6.4 Gráficas de la prueba 1.

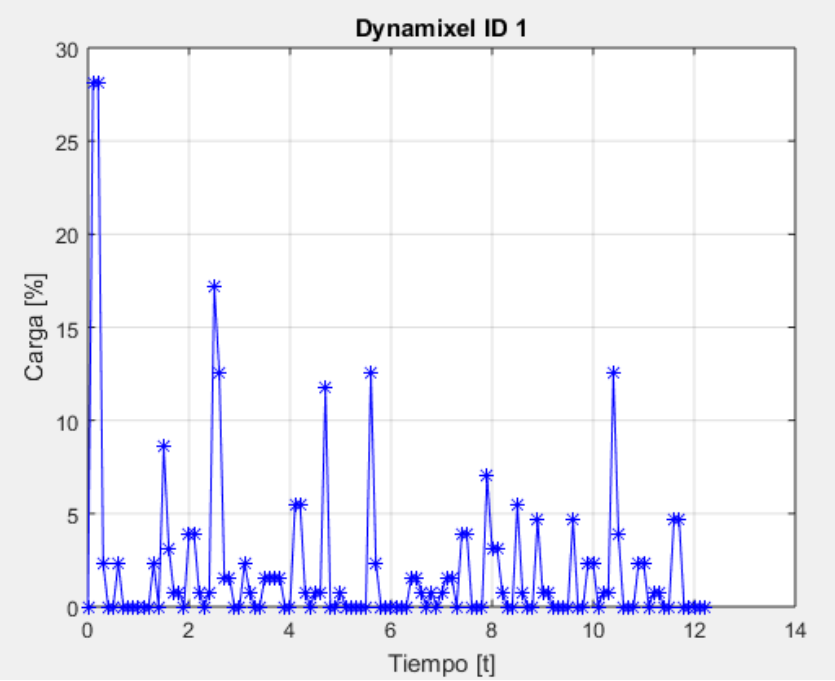
Gráficas del servomotor 1 (base del robot, imagen 3.26).



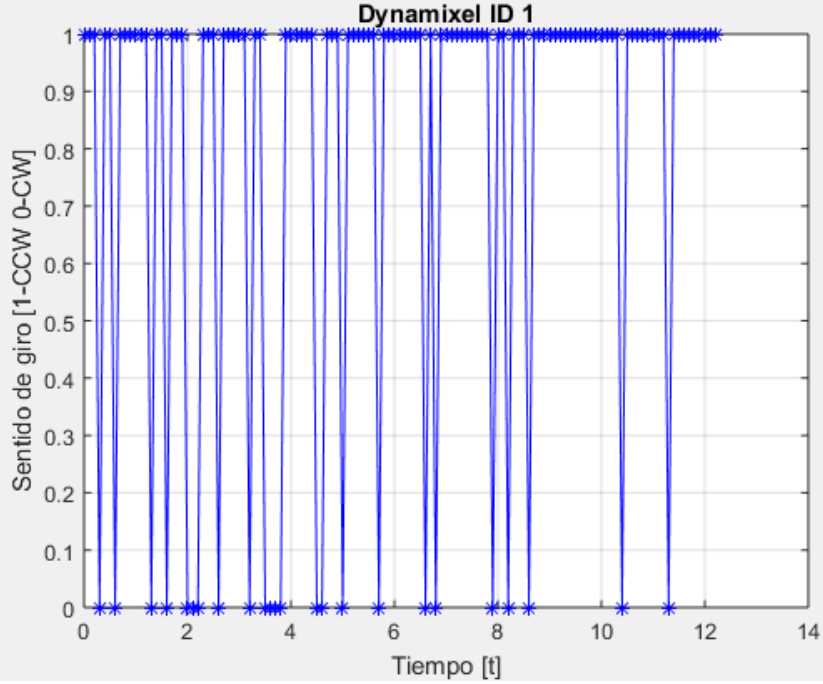
Gráfica 1. Comparación entre la posición deseada y la posición real leída del servomotor.



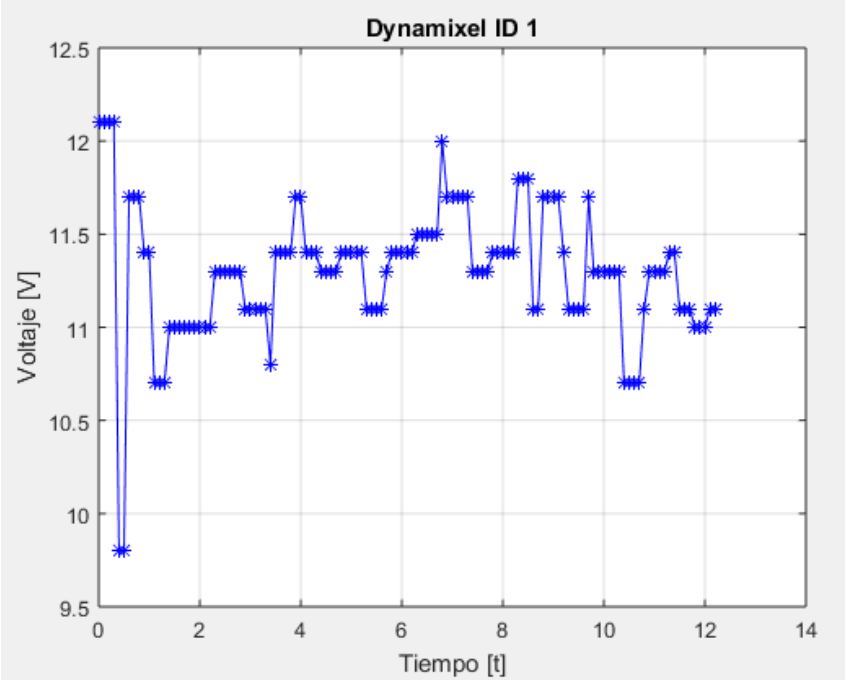
Gráfica 2. Velocidad de desplazamiento del servomotor entre cada movimiento.



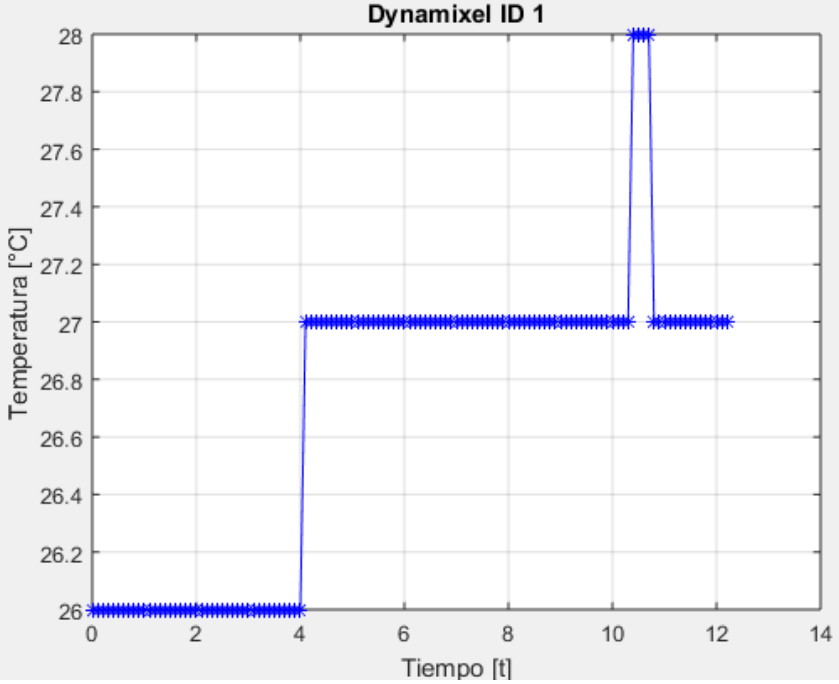
Gráfica 3. Porcentaje de carga que soporta el servomotor en cada instante de la prueba.



Gráfica 4. Sentido de giro del servomotor en cada instante de la prueba (1=sentido anti horario, 0=sentido Horario).

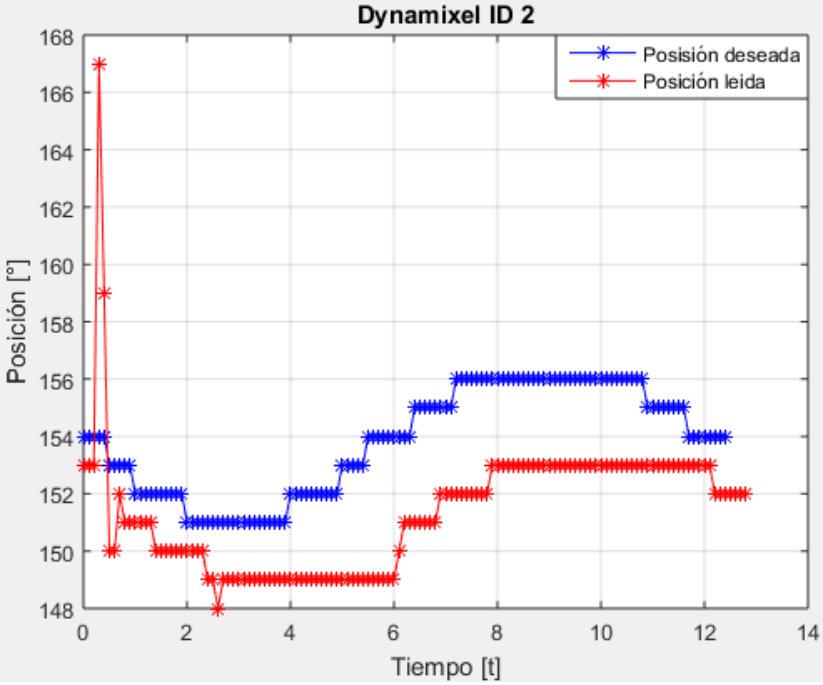


Gráfica 5. Voltaje consumido por el servomotor durante la prueba.

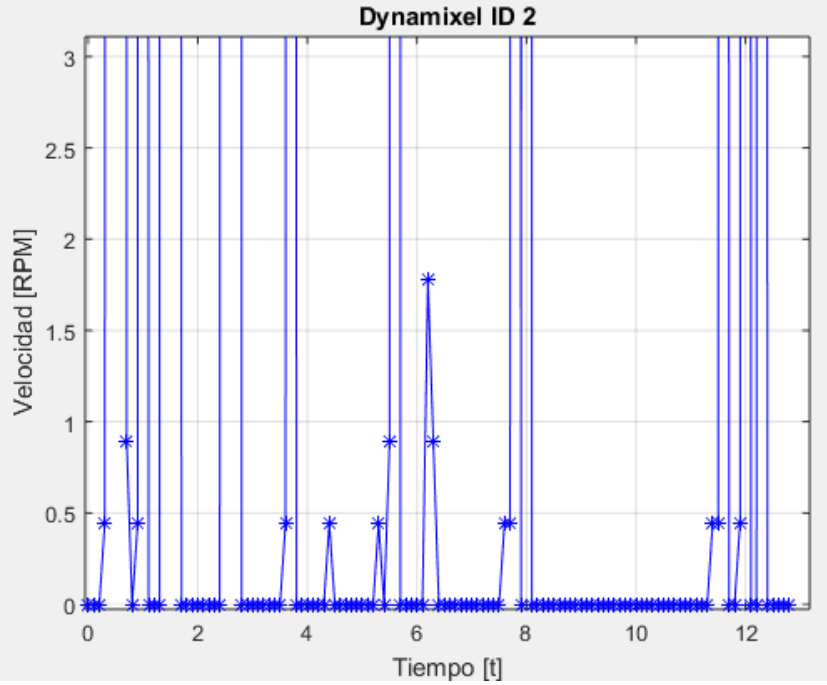


Gráfica 6. Temperatura del servomotor durante la prueba.

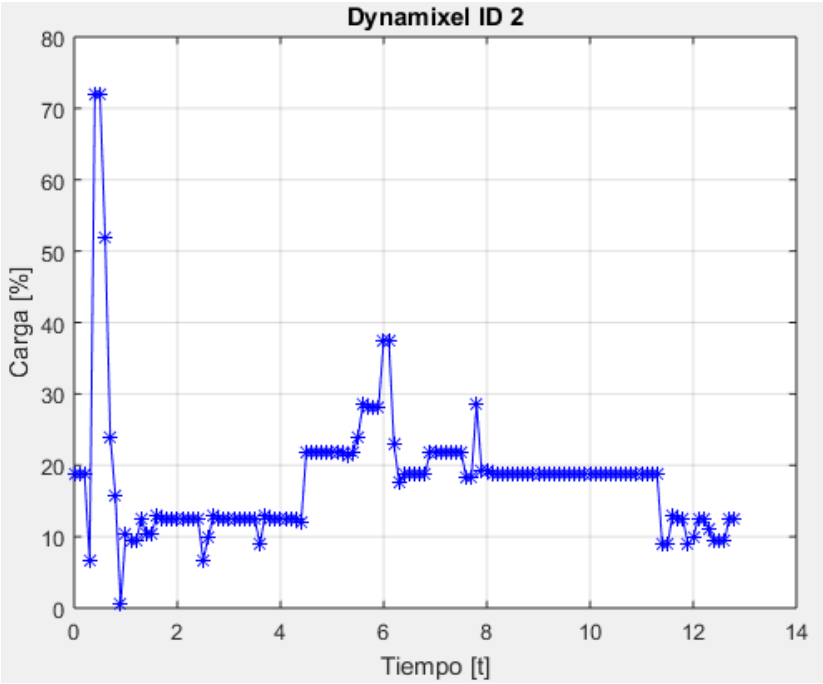
Gráficas del servomotor 2 (imagen 3.26).



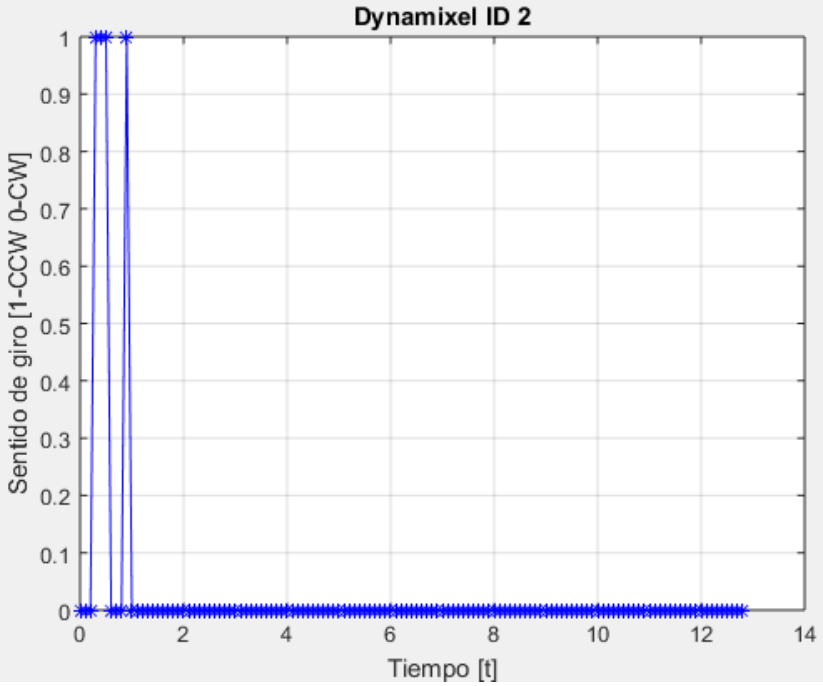
Gráfica 7. Comparación entre la posición deseada y la posición real leída del servomotor.



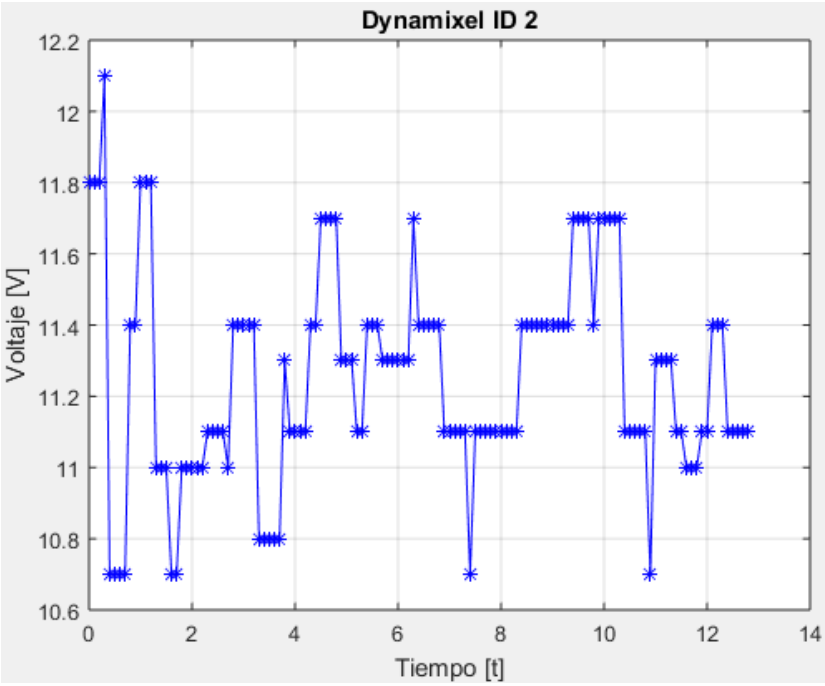
Gráfica 8. Velocidad de desplazamiento del servomotor entre cada movimiento.



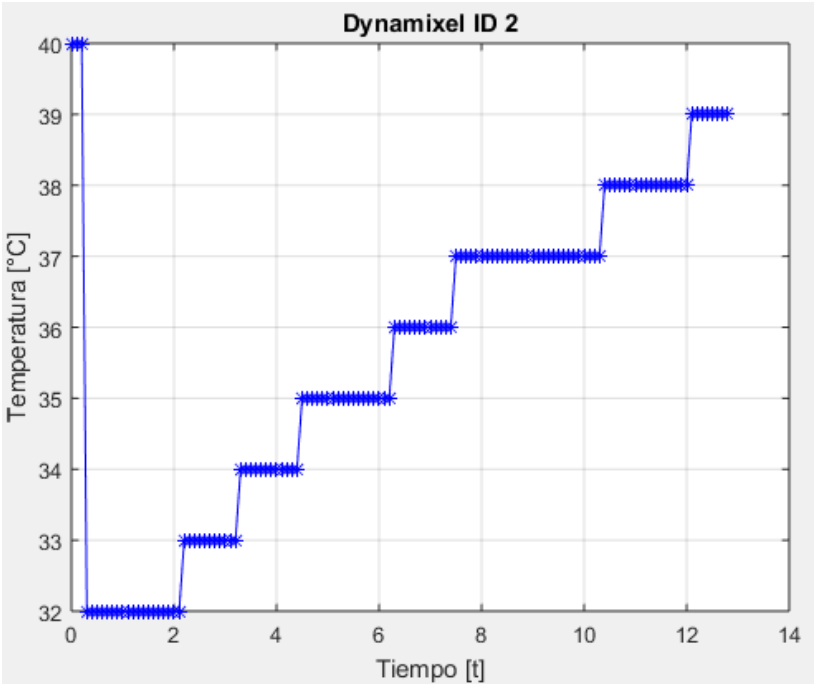
Gráfica 9. Porcentaje de carga que soporta el servomotor en cada instante de la prueba.



Gráfica 10. Sentido de giro del servomotor en cada instante de la prueba (1=sentido anti horario, 0=sentido Horario).

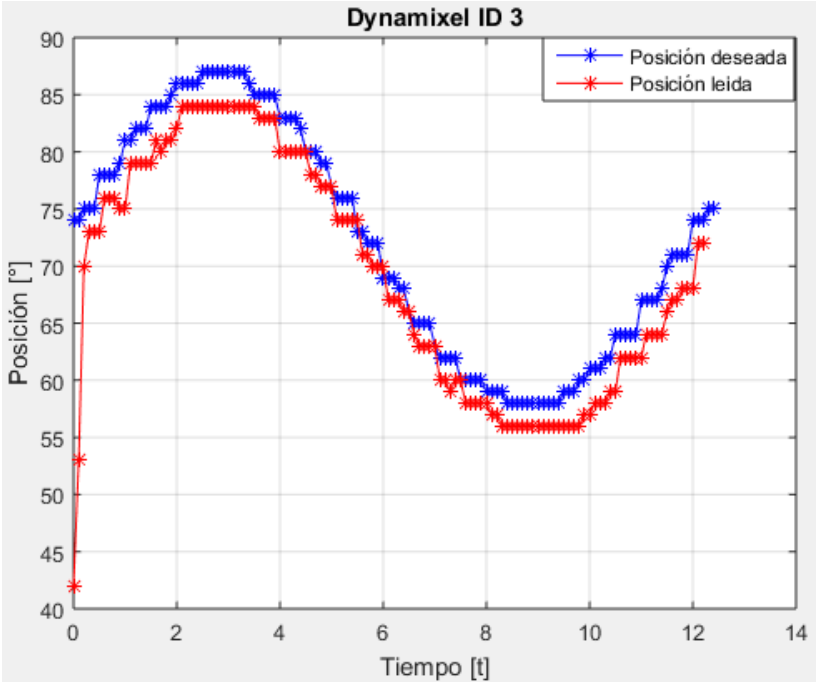


Gráfica 11. Voltaje consumido por el servomotor durante la prueba.

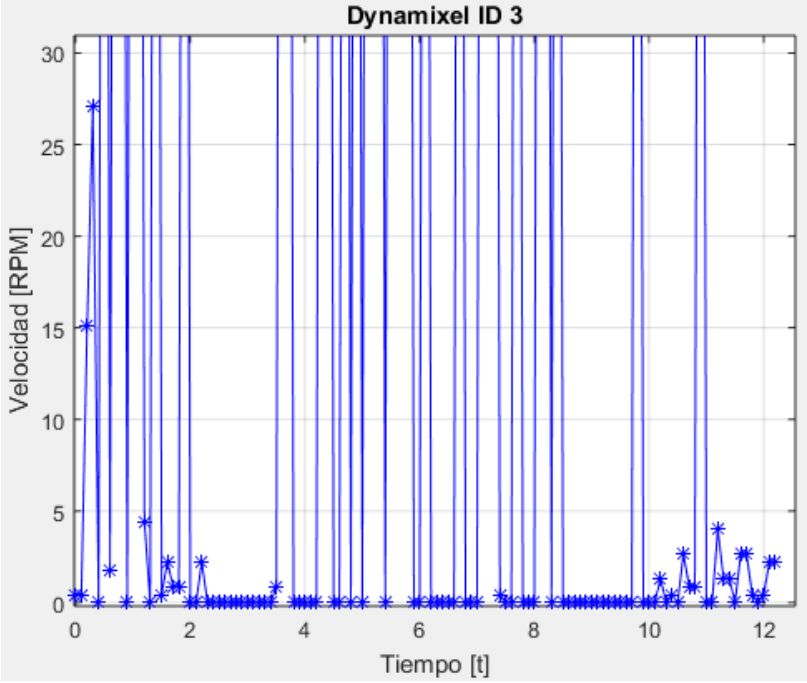


Gráfica 12. Temperatura del servomotor durante la prueba.

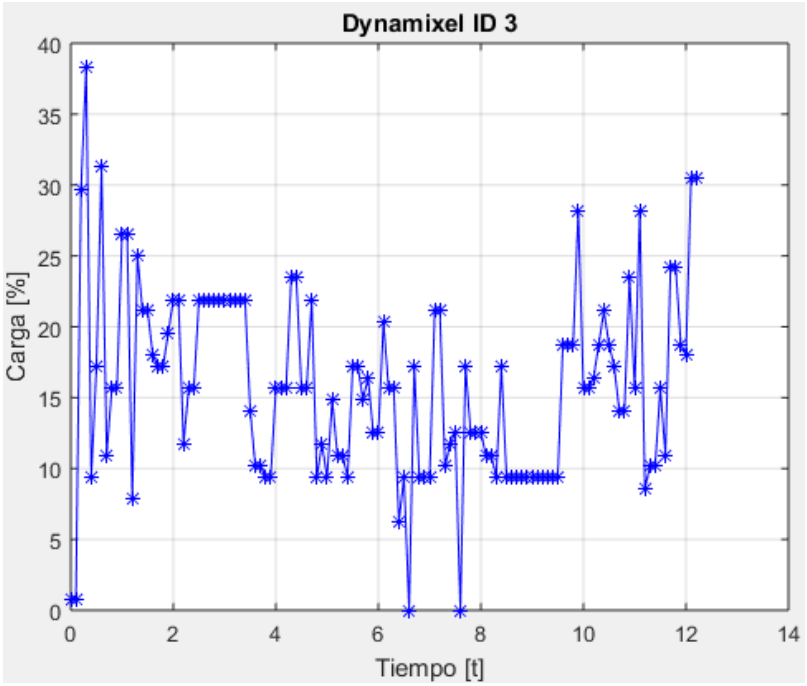
Gráficas del servomotor 3 (imagen 3.26).



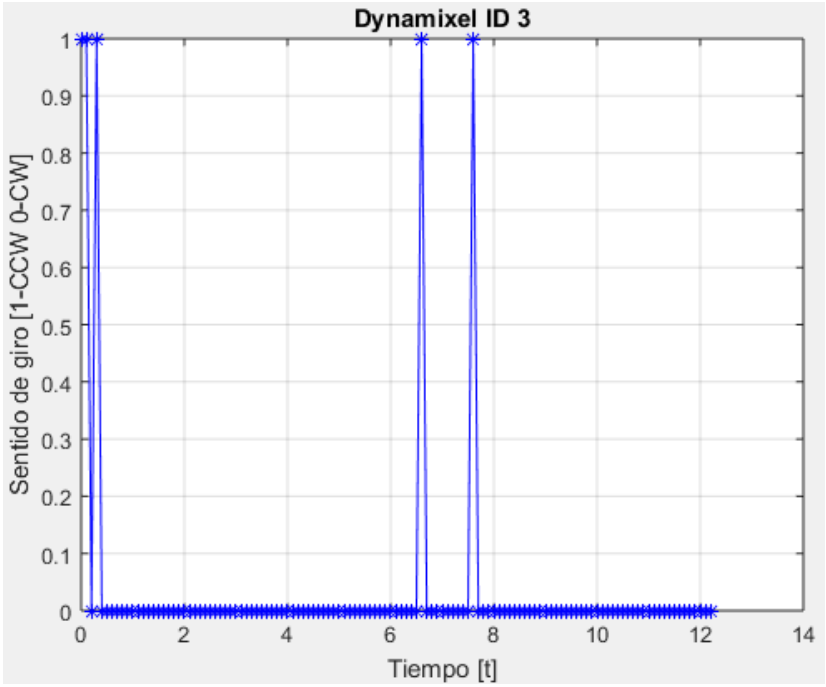
Gráfica 13. Comparación entre la posición deseada y la posición real leída del servomotor.



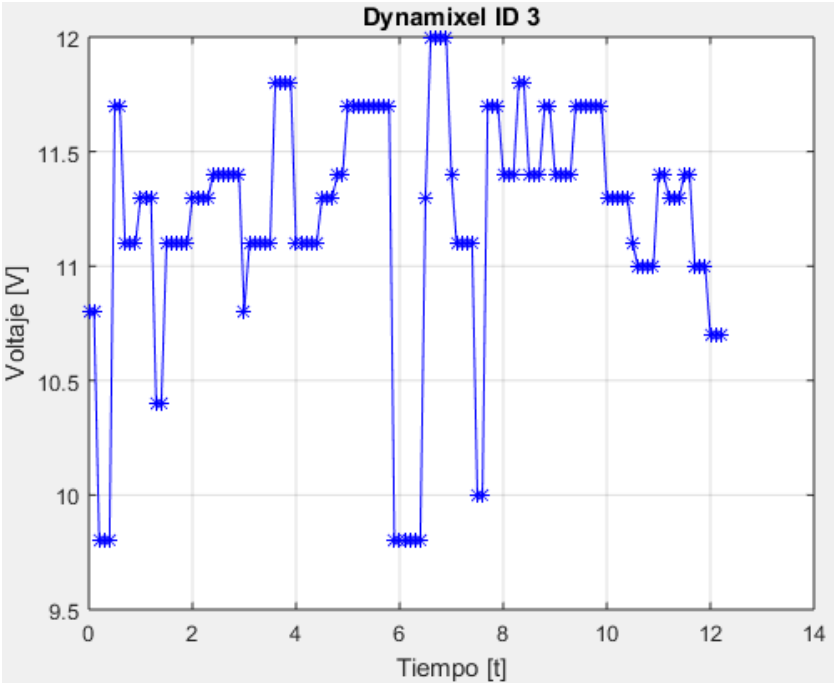
Gráfica 14. Velocidad de desplazamiento del servomotor entre cada movimiento.



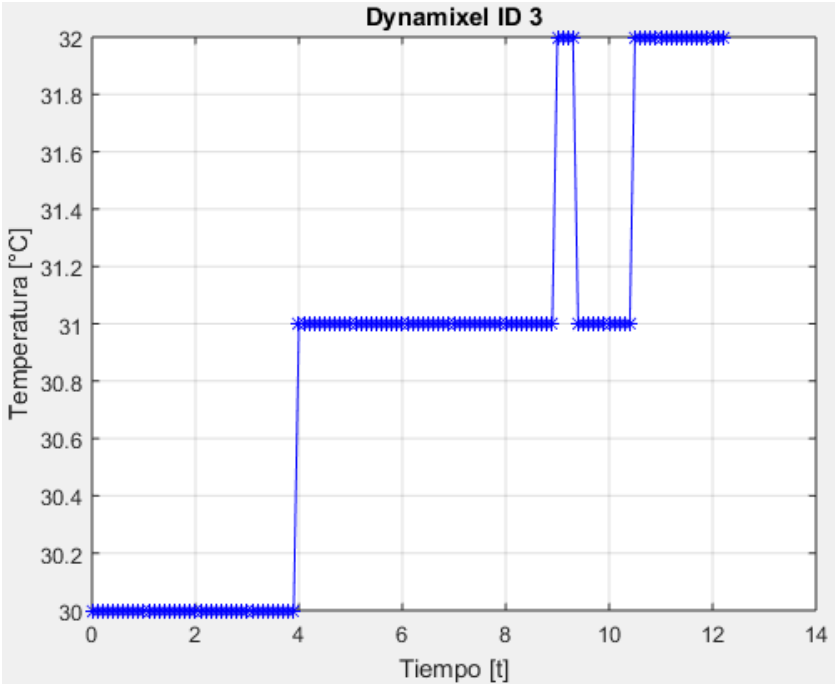
Gráfica 15. Porcentaje de carga que soporta el servomotor en cada instante de la prueba.



Gráfica 16. Sentido de giro del servomotor en cada instante de la prueba (1=sentido anti horario, 0=sentido Horario).



Gráfica 17. Voltaje consumido por el servomotor durante la prueba.



Gráfica 18. Temperatura del servomotor durante la prueba.

6.4 Descripción de la prueba 2 (SIMULINK – MD25):

La prueba consiste en hacer que los motores conectados a la tarjeta MD25 puedan cambiar la velocidad de giro, el sentido de giro o detenerse mediante el uso del módulo SIMULINK - MD25 que se diseñó, para así obtener las mediciones de los encoders asociados a cada uno de los motores, el voltaje de operación en tiempo real y la corriente que consume cada uno de los motores. Es decir, esta prueba consiste en verificar el correcto funcionamiento del módulo SIMULINK - MD25.

Para lograr lo anterior, se mandan 7 valores distintos a cada uno de los motores de manera independiente, es decir, primero se le envian los 7 datos al motor A y posteriormente esos mismos datos al motor B. El tiempo en que se envian los datos son arbitrarios, para poder observar que el valor de los encoders incrementa y decrementa de manera adecuada, además, el conteo no es igual a pesar de que los valores que se le envían son los mismos debido a que el tiempo de operación de los motores no es el mismo. Ambos motores inician la prueba detenidos, en la imagen 6.12 se muestra el módulo que muestra los valores leídos de los motores DC.

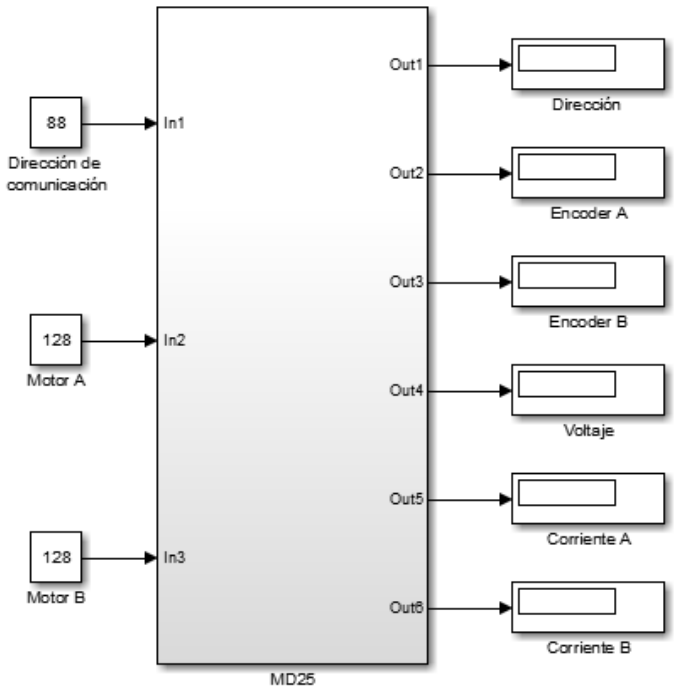


Imagen 6.12 Configuración en SIMULINK para realizar la Prueba 2.

6.5 Ejecución de la prueba 2

Los parámetros que se utilizaron son los que se aprecian en las tablas 6.4 y 6.5.

Tabla 6.4. Valores de velocidades al motor A.

Motor	Velocidad	Descripción
A	128	Motor Detenido
	190	50% Velocidad adelante
	255	100% Velocidad adelante
	128	Motor detenido
	65	50% Velocidad atrás
	0	100 % Velocidad atrás
	128	Motor detenido

Tabla 6.4. Valores de velocidades al motor B.

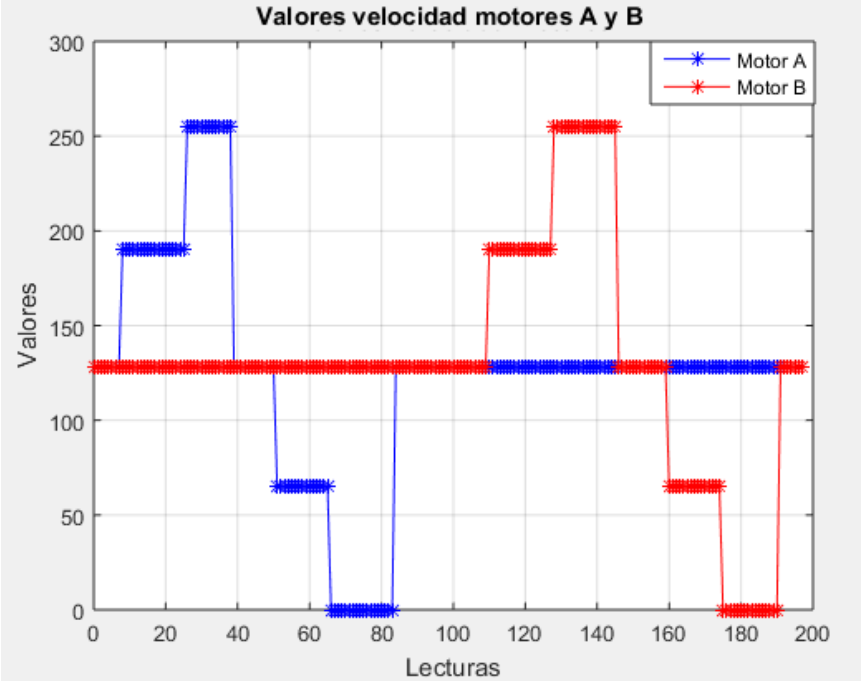
Motor	Velocidad	Descripción
B	128	Motor Detenido
	190	50% Velocidad adelante
	255	100% Velocidad adelante
	128	Motor detenido
	65	50% Velocidad atrás
	0	100 % Velocidad atrás
	128	Motor detenido

Una vez terminada la prueba, se guardan los datos enviados y leídos de a MD25 en el Workspace de MATLAB, esos datos se almacenan en una hoja de cálculo para posteriormente procesarlos con un script en MATLAB, llamado Graficas_MD25, el cual genera 6 gráficas:

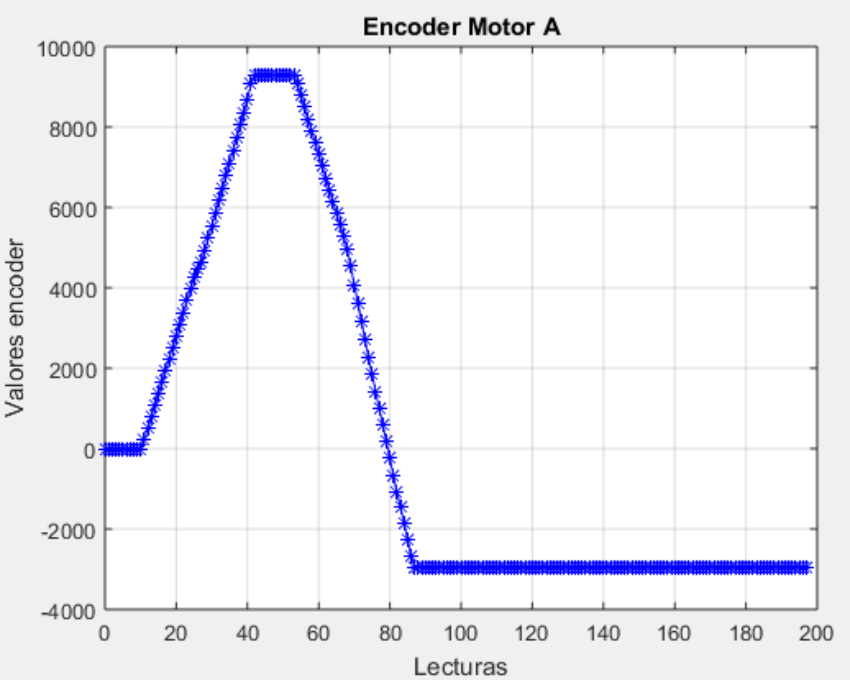
1. Valores de velocidad motores A y B (Valores enviados a la MD25)
2. Encoder asociado al motor A
3. Encoder asociado al motor B
4. Voltaje operación MD25
5. Corriente motor A
6. Corriente motor B

El script se puede encontrar en el Apéndice 10 (Script para generar gráficas)

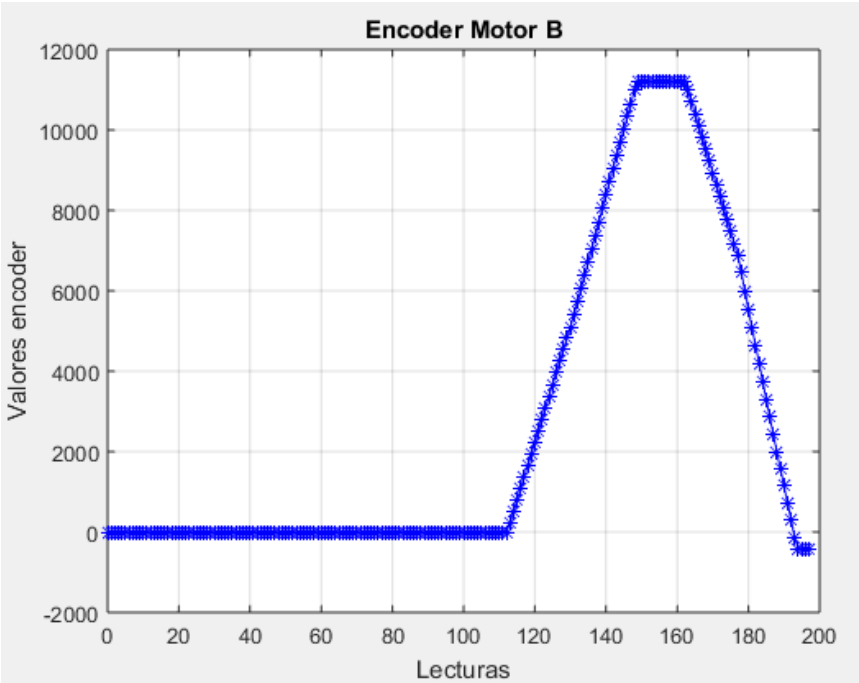
6.6 Gráficas de la Prueba 2.



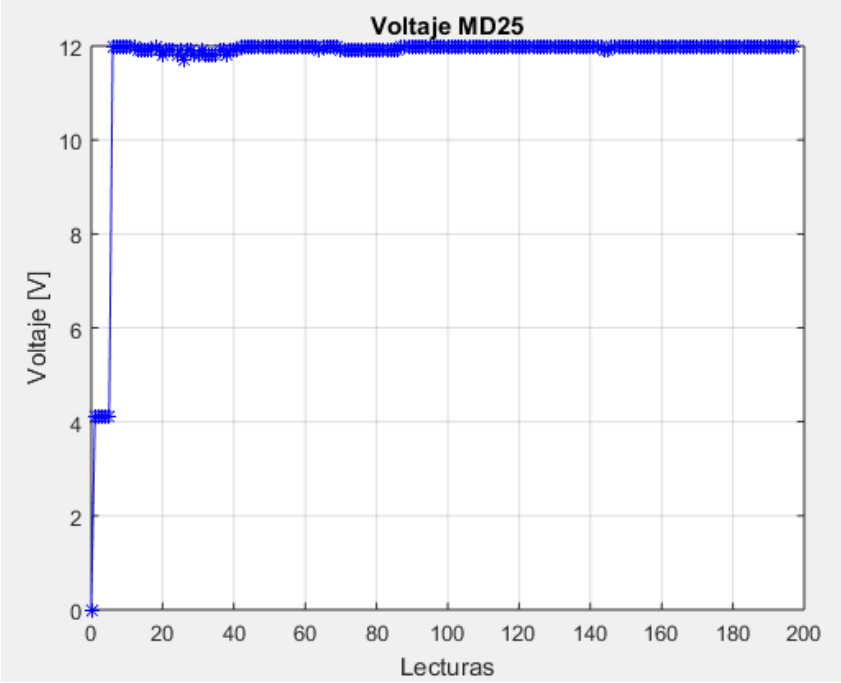
Gráfica 19. Comparación de la velocidad entre los motores A y B durante la prueba.



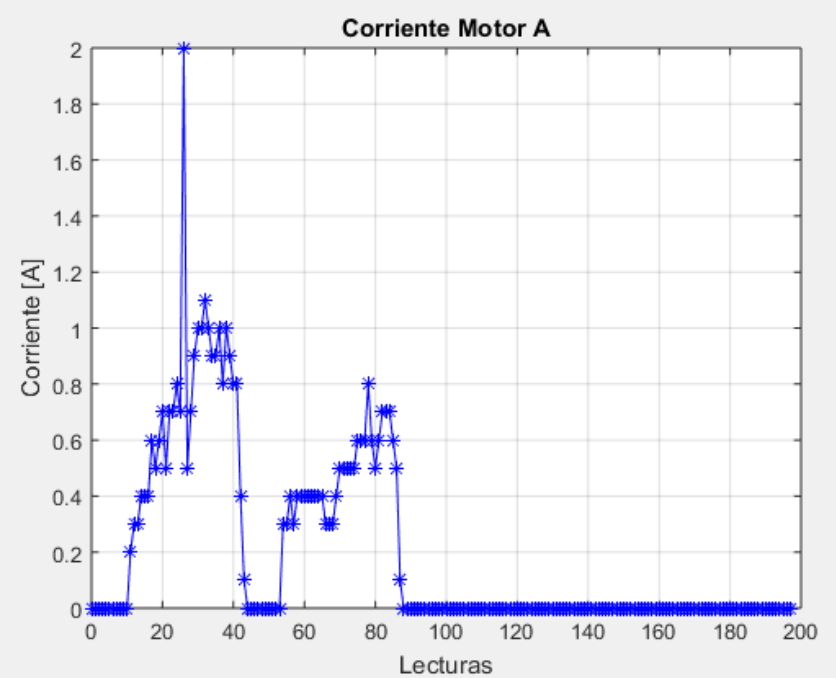
Gráfica 20. Valores leídos del encoder del motor A durante la prueba.



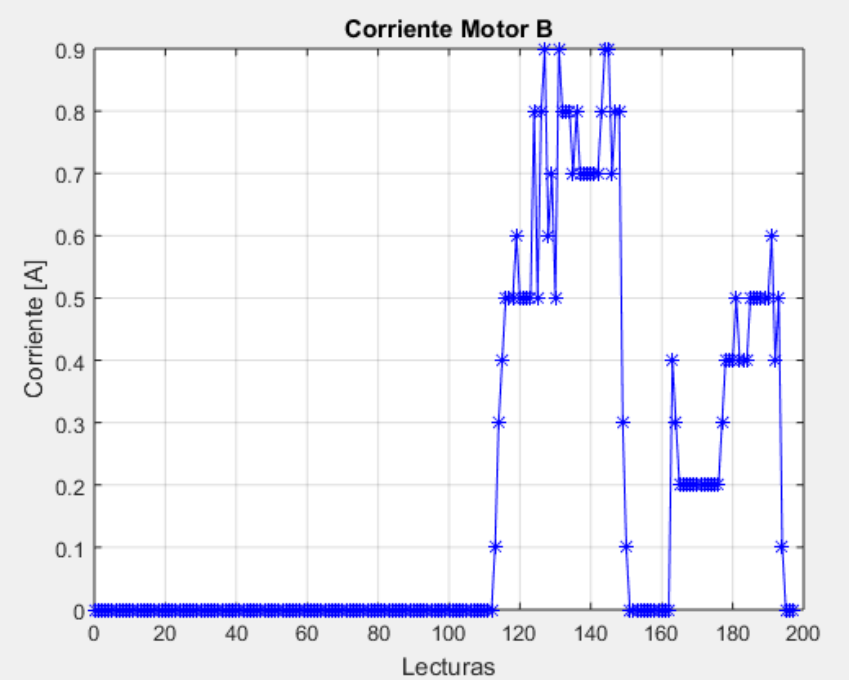
Gráfica 21. Valores leídos del encoder del motor B durante la prueba.



Gráfica 22. Voltaje consumido por la tarjeta MD25 durante la prueba.



Gráfica 23. Corriente consumida por el motor A durante la prueba.



Gráfica 24. Corriente consumida por el motor B durante la prueba.

Capítulo 7

Conclusiones y trabajo a futuro

7.1 Conclusiones

En ésta tesis se logró hacer el diseño y la construcción del prototipo funcional de un robot adaptable, también se lograron diseñar herramientas de forma modular, que permiten controlar al robot adaptable y leer los valores de voltaje, temperatura, corriente, posición, velocidad y carga de los servomotores Dynamixel. Además se logró obtener un módulo para la tarjeta MD25 que permite controlar la velocidad y sentido de giro de motores DC que se conecten a la tarjeta Md25 (Motores EMG30, Micro-motorreductores Pololu, etc.) y leer los valores de los encoders, voltaje de operación y corriente de los motores. Con lo anterior, se logró obtener un banco de pruebas para un robot adaptable, cumpliendo los objetivos propuestos al principio de ésta tesis.

En el diseño del robot adaptable fue necesario tener presente las propiedades mecánicas, limitaciones y alcances de los servomotores Dynamixel Rx24 y Rx28, para poder obtener un análisis de cargas aproximado y así determinar si los servomotores a utilizar son los adecuados. Cada servomotor Dynamixel Rx24 y Rx28 tienen un límite de carga de 26 $Kg \cdot cm$ y 37.7 $Kg \cdot cm$ respectivamente. Como se vio en el capítulo 3, el análisis estático del robot adaptable dio como resultado que los servomotores soportarían sin problema las cargas de los eslabones, sobre todo en su servomotor 2 que es el más crítico (imágenes 3.26 y 3.27). En la práctica, el servomotor 2 (imágenes 3.26 y 3.27) que soporta a los dos eslabones comienza a presentar problemas al operar con esa carga, aunque no debería de suceder, pues los cálculos arrojaron que la carga máxima que soporta éste en su configuración más crítica, es decir, los dos eslabones extendidos y a un ángulo perpendicular al servomotor (90°), la carga es de 14.597 $Kg \cdot cm$, y el servomotor puede llegar a soportar hasta una carga de 37.7 $Kg \cdot cm$.Lo anterior puede justificarse porque el valor que proporciona el fabricante de carga que soportan dichos servomotores está dado con base en cargas estáticas y no nos proporcionan el valor que éstos pueden llegar a soportar al operar bajo cargas dinámicas, como en las que opera el robot adaptable. Como consecuencia, esto genera que los servomotores demanden más corriente y a su vez se traduzca en un aumento considerable en la temperatura de operación de los mismos, haciendo que se active el mecanismo de seguridad de temperatura interno con el que cuentan los servomotores y haga que el sistema falle en su punto más crítico.

Otra posible justificación de que los servomotores presenten problemas al soportar las cargas del robot, es debido a la alimentación eléctrica con la que se cuenta de los mismos, ya que éstos están conectados internamente en serie, generando que cuando uno presenta problemas, provoque que los demás fallen, esto se podría solucionar conectando cada servomotor a una

fuente independiente para que de ésta manera, los servomotores puedan demandar la corriente necesaria para que operen correctamente. Es posible desactivar el sistema de seguridad con el que cuentan, pero no se recomienda precisamente porque se pueden dañar irreparablemente, dejándolos inservibles. Por lo anterior, una vez que se active el sistema de alarma de seguridad para los servomotores, se debe dejar reposar 20 minutos para poder volver a utilizar todo el sistema.

Con respecto al mecanismo diseñado del robot adaptable, se logó hacer que éste aprovechara al máximo los ángulos de operación en los cuales operan los servomotores. Se logró que el servomotor de la base que hace girar a los eslabones alrededor de la misma, se desplazara sobre su rango preestablecido de operación estándar que va de 0 a 300 grados. El diseño de los eslabones, permitió que los ángulos de operación de los otros 2 servomotores fuera de 60 a 240 grados. En estos se pierden 120 grados, para evitar que los eslabones choquen con otro servomotor, con otro eslabón o con la misma base que soporta a todo el mecanismo.

La estructura del prototipo del robot adaptable, fue ensamblado y pegado con pegamento para acrílico, lo que ayudó a que las piezas quedaran unidas como si fueran una sola pieza, gracias a que el pegamento funde el acrílico permitiendo que se adhieran correctamente.

La simplicidad del diseño de los elementos de la estructura mecánica del robot hizo que ensamble fuera fácil de realizar, como lo estipulado en el CAD. En el ensamble físico se tuvieron algunas variaciones en las dimensiones de las piezas, pues no se tenía una manera de asegurar que algunas de éstas estuvieran correctamente alineadas y en otros casos éstas tuvieron que ser modificadas pues presentaban errores provocados por algunas imprecisiones en el corte laser de las piezas, pues se corroboró que algunas tenían variaciones considerables, en cuanto a que tenían medidas diferentes a otras, siendo que habían sido cortadas en la misma máquina y habían estado en la misma placa de acrílico. Ciertos cortes no fueron hechos de manera trasversal en su totalidad, sino que presentaban algunas curvaturas en ciertas caras, provocando que algunas piezas no estuvieran completamente planas y por tal motivo no quedaran alineadas con la cara a unir. El robot adaptable se ensambló como se había esperado, a pesar de las dificultades antes mencionadas, lo que garantiza su funcionamiento como se esperaba.

Cuando el robot adaptable está trabajando con un sólo eslabón adaptable, la parte interior del mismo, que es el que se desplaza hacia afuera o hacia adentro con respecto a la parte exterior del eslabón adaptable, funciona de manera correcta, ya sea mientras está contraído o extendido, pero cuando al robot adaptable se le agrega un eslabón más, debido a su cualidad de ser modular, la parte interior del primer eslabón adaptable empieza a tener un comportamiento imprevisto y que sólo se presenta cuando éste se encuentra en su punto más alargado (extendiendo). Dicho movimiento genera que el arreglo presente una pequeña vibración, que puede llegar a causar problemas en un futuro, ya que se tienen piezas de

acrílico que se pueden desgastar con facilidad mediante el uso continuo y provocar que se lleguen a presentar imprecisiones durante su funcionamiento.

En el diseño del software, el método iterativo permitió tener distintas versiones de los módulos diseñados y construidos en SIMULINK para controlar los elementos que conforman el robot adaptable. La primera versión permitió establecer la comunicación desde SIMULINK con Arduino enviando información máxima de 8 bits, la siguiente versión permitió enviar información mayor de 8 bits (Separando los números en bytes), las versiones posteriores además de enviar información permitieron recibir información desde Arduino en SIMULINK.

La versión final a la que se llegó en este trabajo de los módulos de SIMULINK y los programas de los microcontroladores permiten enviar y recibir información, la cual necesita ser pre-procesada y post-procesada la información, dependiendo de quién sea el emisor y quien sea el receptor (SIMULINK a Arduino o Arduino a SIMULINK).

El haber hecho una analogía con los elementos del robot adaptable con el proceso de la comunicación fue de gran importancia, ya que se realizó un algoritmo que indicara quien sería el emisor y quien el receptor del mensaje de las instrucciones al robot y de las lecturas de los sensores. Primero la información se envía desde SIMULINK (emisor) a Arduino (Receptor), por lo cual SIMULINK necesita pre-procesar la información (mensaje), para poderla enviar al microcontrolador (canal), cuando Arduino recibe la información, necesita post-procesarla para comunicarse de manera adecuada con los elementos a controlar. Para las lecturas de las variables se requiere un proceso invertido, ya que en este caso el microcontrolador seria el emisor y SIMULINK el receptor. Por lo anterior, el algoritmo es de gran importancia, ya que permite cerrar el círculo de comunicación (SIMULINK-Arduino) y que el mensaje (información) llegue de manera adecuada desde el emisor al receptor. Además, la forma de comunicación que se realizó en este proyecto entre SIMULINK y Arduino, no sólo funciona para los servomotores Dynamixel y la tarjeta MD25, ya que se diseñó de manera general y es posible utilizar cualquier otro microcontrolador o dispositivo que utilice comunicación serial, lo cual es de gran importancia porque podría utilizarse en diversos proyectos.

Otra ventaja de las herramientas creadas en SIMULINK es que están diseñadas de una forma modular, lo que permite impulsar múltiples funcionalidades y su reutilización, esto se logra al generar nuevos usos o agregarle más bloques de SIMULINK que permitan potencializar la capacidad de los módulos creados.

Una ventaja de los módulos diseñados es que se pueden llegar a controlar hasta 255 servomotores con el módulo SIMULINK-Dynamixel y hasta 8 tarjetas MD25 Con el módulo SIMULINK-MD25. Esta ventaja es importante en aplicaciones donde no se requiera enviar y recibir datos de todos los elementos al mismo tiempo o en un rango mínimo de tiempo,

debido a que se tiene una conexión en serie entre los elementos y sólo se puede enviar y recibir información elemento por elemento, lo que representa una desventaja en el banco de pruebas del robot adaptable, ya que lo que se busca es obtener los valores de todos sus elementos en tiempo real para no perder información.

En esta primera versión del banco de pruebas se diseñó un selector que permite enviar y recibir datos en un tiempo determinado para cada uno de sus elementos, con lo que fue posible medir las variables de interés, anteriormente descritas, obteniendo así las características de funcionamiento del robot adaptable. Sin embargo, en lugar de utilizar un selector, se propone utilizar un módulo de los diseñados para cada elemento, obteniendo una conexión en paralelo que permita enviar y recibir información de todos sus componentes en tiempo real. Lo anterior requeriría un puerto serial distinto por cada bloque que se utilice.

Otra ventaja de haber empleado una metodología de diseño en el desarrollo de software, permitió generar un software funcional y que además es posible mejorarlo continuamente, según se requiera, para que su alcance sea cada vez mayor.

Una de las características que se puede considerar para un *mantenimiento de adaptación* (modificaciones que se necesitan realizar por un cambio externo al sistema) de los módulos de SIMULINK, es que tengan la capacidad de utilizarse tanto el módulo de Dynamixel como el módulo de las Tarjetas MD25 con un sólo microcontrolador.

Al tener una herramienta en SIMULINK de propósito general resultaba complicado que el robot adaptable siguiera una trayectoria y proporcionara las mediciones, por lo cual fue necesario hacer un *mantenimiento de adaptación* (Como se realizó en la prueba 1, capitulo 6).

Independientemente de las pruebas realizadas y documentadas en ésta tesis, cabe mencionar que se realizaron otras dos pruebas que arrojaran valores del comportamiento del robot, cuando éste está ejecutando la misma trayectoria pero con dos configuraciones diferentes. Dichas pruebas se hicieron con una configuración en la que los eslabones del robot adaptable se encuentran contraídos y otra cuando los eslabones se encuentran extendidos (imágenes 3.26 y 3.27 respectivamente), obteniendo así que cuando se tiene una tarea para realizar con dos diferentes configuraciones del robot adaptable, existen variaciones considerables en el comportamiento de los actuadores, ya que cuando el robot hizo la trayectoria de la elipse en su estado inicial (los eslabones con l_1=18 cm y l_2=14 cm) se tienen lecturas de voltaje que los actuadores consumen entre 12 V y 11.7 V, el porcentaje de carga está entre 6% y 7% y la temperatura se encuentra entre 28°C y 43°C. En comparación con la segunda configuración del robot adaptable, al seguir la misma trayectoria de la elipse, pero esta vez con sus eslabones completamente extendidos (l_1=22 cm y l_2=18 cm), se obtienen lecturas de voltaje que los actuadores consumen entre 11.3 V y 10.8 V, el porcentaje de carga está entre 15% y 19% y la temperatura se encuentra entre 29°C y 49°C. De lo anterior es posible decir que

dependiendo de la configuración que se tenga en el robot adaptable, para realizar una tarea, ésta afectará directamente a los actuadores. Como se vio en el voltaje, temperatura y porcentaje de carga, son parámetros que en teoría no tendrían por qué moverse de un valor de 12 V, 6 o 7% de carga y una temperatura de entre 28°C y 40 °C, pues son en esos valores donde los actuadores operan con normalidad, al sobre pasarlos provoca que no se tenga un correcto funcionamiento del robot adaptable, provocando que se sobrecalienten los motores, generando que éste se detenga cuando realiza una tarea.

7.2 Trabajo a futuro

En un futuro, el robot podría ser fabricado en otro tipo de material distinto al acrílico, por ejemplo: aluminio o algún tipo de acero, con la finalidad de que se pueda tener una estructura más firme y resiste a desgastes por uso prolongado y continuo, además, sería posible llevar un mejor control al momento de maquinar las piezas. Como consecuencia de cambiar el material del que esté construido el robot adaptable, se tendría que utilizar unos motores y servomotores con mayor capacidad de carga, pues el peso del robot aumentaría considerablemente.

Una solución para que los servomotores soporten las cargas producidas por los eslabones del robot adaptable, como parte de trabajo a futuro, es el hacer que cada servomotor se alimente de una fuente distinta para energizar a estos, pues al estar todos conectados en la misma fuente de alimentación, provoca que ésta no suministre la demanda de corriente necesaria para cada servomotor, lo que genera que en ciertas posiciones los servomotores no soporten la carga de los eslabones. Otra solución podría ser el cambiar los servomotores por otros, que sean capaces de soportar mayor carga, pero ésta solución sería más costosa, lo que la convierte en una opción no conveniente.

Actualmente el robot cuenta con unos finales de carrera (micro-switches) colocados sobre una pieza de acrílico que está sujeta por afuera de la parte exterior del eslabón adaptable, en un futuro estos podrían ser sustituidos por algún otro tipo de sensor, que permita delimitar el espacio de trabajo por donde se desplaza la parte interior del eslabón adaptable, con la finalidad de hacer que el robot tenga una mejor estética y de que los sensores puedan medir con una mayor precisión.

La estructura mecánica del robot adaptable, puede ser mejorada en un futuro, con la finalidad de que funcione mejor y mejorar su estética, además que los eslabones adaptables tengan un mayor rango de elongación.

Referencias

(UNSAAC), Universidad Nacional de San Antonio Abad del Cusco. (2006). Robótica. Retrieved 16 Mayo, Mayo 2016, from https://robotica.wordpress.com/about/

Acrilico, Ultra Plas S.A. de C.V. y el. (2016). Propiedades del acrílico. Retrieved 23 Enero, 2016, from http://www.acrilico-y-policarbonato.com/acrilico-propiedades.html

Albano, Luis. (2012). Cola de milano. Retrieved Enero 20, 2016, from https://micarpinteria.wordpress.com/2012/04/09/cola-de-milano-dovetail/

Albano, Luis. (2016). Mi carpinteria, apuntes de carpinteria: aprende, crea, comparte. Retrieved 17 de Mayo, 2016, from https://micarpinteria.wordpress.com/2012/04/15/caja-y-espiga/

Aracil, Rafael, Balaguer, Carlos, & Armada, Manuel. (2008). Robots de servicio. *Revista Iberoamericana de Automática e Informática Industrial RIAI, 5*(2), 6-13.

Arduino. (2016a). Arduino Mega. Retrieved 15 Octubre, 2015, from https://www.arduino.cc/en/Main/arduinoBoardMega

Arduino. (2016b). attachInterrupt(). Retrieved 30 Marzo, 2015, from https://www.arduino.cc/en/Reference/AttachInterrupt

Arduino. (2016c). Comunicando Arduino con otros sistemas. Retrieved 16 Febrero, 2016, from http://playground.arduino.cc/ArduinoNotebookTraduccion/Appendix4

Arduino. (2016d). Write Library. Retrieved 15 Febrero, 2015, from https://www.arduino.cc/en/Reference/Wire

Barrientos, Antonio. (2002). Nuevas aplicaciones de la robótica. Robots de servicio. *Avances en robótica y visión por computador. Cuenca, Ediciones Castilla-La Mancha, 288*.

Barrientos, Antonio, Peñín, Luis Felipe, Balaguer, Carlos, & Aracil, Rafael. (2007). *Fundamentos de robótica*: McGraw-Hill, Interamericana de España.

Baturone, Aníbal Ollero. (2005). *Robótica: manipuladores y robots móviles*: Marcombo.

Birch, Malim Tony, Ann, Hayward, & Sheila, Gutiérrez Manzanilla. (1998). *Psicología comparada: conducta humana y animal: un enfoque sociobiológico*.

Constructora, Azteca. (2016). Bancos de pruebas. Retrieved 5 Mayo, 2016, from http://www.ccazteca.com.mx/equipos/bancos-de-pruebas

Costas, Matías Romero. (2012, 2012). Robótica: entra al mundo de la inteligencia artificial. *+conectad@s*, 32.

Electronics, Robot. (2011a). MD25- Dual 12 volts 2.8 Amp. H Bridge Motor Drive Documentation. Retrieved 15 Mayo, 2015, from http://www.robot-electronics.co.uk/htm/md25i2c.htm

Electronics, Robot. (2011b). MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive Retrieved Diciembre, 2015, from MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive

Electronics, Robot. (2011c). MD25 RD02 Motor Controller I2C. Retrieved 18 Febrero, 2015, from http://www.robot-electronics.co.uk/htm/arduino_examples.htm#MD25%20RD02%20Motor%20Controller

Electronics, Savage. (2011). Arduino y dynamixel ax12. Retrieved 25 Febrero, 2015, from http://savageelectronics.blogspot.mx/2011/01/arduino-y-dynamixel-ax-12.htm

Fuller, James L. (1991). Robotics Introduction, Programming, ond Project. Coller Toronto: Macmillan Canada: Inc.

González, Víctor R. (2004). Robots industriales. Retrieved 9 Mayo 2016, from http://platea.pntic.mec.es/vgonzale/cyr_0708/archivos/_15/Tema_5.4.htm

Jesús. (2014). Comunicacion I2C. Retrieved 28 Agosto, 2015, from https://ardubasic.wordpress.com/2014/07/28/comunicacion-i%C2%B2c/

Kelly, Rafael, & Santibáñez, Víctor. (2003). *Control de movimiento de robots manipuladores*: Pearson Educación, SA.

Llamas, Luis. (2014). Comunicación de arduino con puerto serie. Retrieved 19 Noviembre, 2015, from http://www.luisllamas.es/2014/04/arduino-puerto-serie/

MathWorks. (2016). Numeric Types. Retrieved 23 Abril, 2016, from http://es.mathworks.com/help/matlab/numeric-types.html

MATLAB. (2013). MATLAB y SIMULINK. Retrieved 25 Octubre, 2015, from http://www.multion.com.mx/micrositios/matlab/inicio.html#descripción

Mecatronics, Naylamp. (2015). Módulo Rs485 Retrieved 15 Octubre, 2015, from http://www.naylampmechatronics.com/alambrico/62-modulo-rs485.html

Pérez, Ing. Eric López.). Ingenieria en microcontroladores, protocolo RS 485. Retrieved 20 Febrero, 2016, from http://www.electronica60norte.com/mwfls/pdf/rs-485.pdf

Prieto, José Mariscal. (2003, 30 Marzo). Robots modulares. Robots reconfigurables por si mismos. Retrieved 16 Marzo, 2015, from http://www.josemariscal.com/downloads/universidad/robotica/robotica.pdf

PROMETEC. (2016). El bus I2C. Retrieved 15 Febrero, 2016, from http://www.prometec.net/bus-i2c/

ROBOTIS. (2009a). Plano de la brida HN07-I101. Retrieved 20 Agosto, 2015, from http://www.robotis.com/view/HN07-i101/HN07-i101.pdf

ROBOTIS. (2009b). Plano de la brida HN07-N101. Retrieved 20 Agosto, 2015, from http://www.robotis.com/view/HN07-N101/HN07-N101.pdf

ROBOTIS. (2009c). Plano del bracket FR07- S101. Retrieved 20 Agosto, 2015, from http://www.robotis.com/view/FR07-S101/FR07-S101.pdf

ROBOTIS. (2009d). Plano del bracket FR07-H101. Retrieved 20 Agosto, 2015, from http://www.robotis.com/view/FR07-H101/FR07-H101.pdf

ROBOTIS. (2010a). Documentación del Dynamixel RX-24F. Retrieved 20 Agosto, 2015, from http://support.robotis.com/en/product/dynamixel/rx_series/rx-24f.htm

ROBOTIS. (2010b). USB2Dynamixel. Retrieved 20 Mayo, 2015, from http://support.robotis.com/en/product/auxdevice/interface/usb2dxl_manual.htm

Saha, Subir Kumar. (2010). *Introducción a la Robótica*. México: McGraw-Hill Interamericana.

Sommerville, Ian, & Galipienso, María Isabel Alfonso. (2005). *Ingeniería del software* (Séptima ed.). Madrid: Pearson Educación.

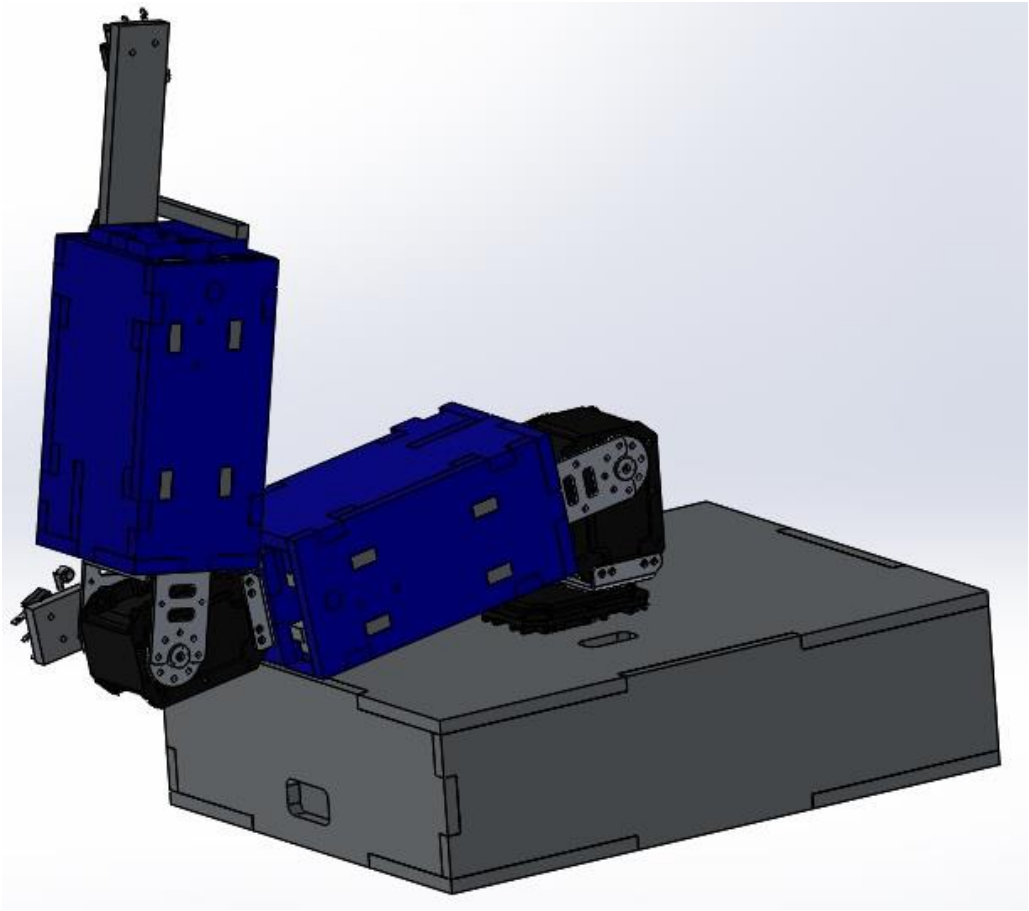
UAH, Grupo de Robótica Educativa. (2000). Protocolo de comunicacion I2C. Retrieved 29 Febrero, 2016, from www.roboticaeducativa.org/mod/resource/view.php?id=1142

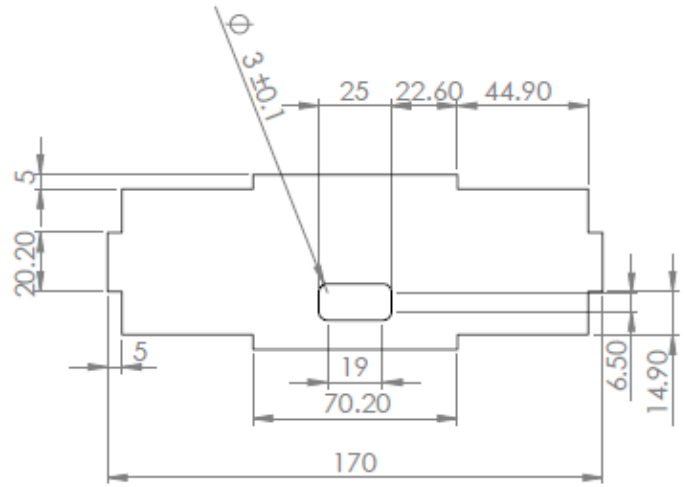
Apéndices

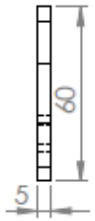
1. Planos del robot adaptable
2. Plano del Micro-motorreductor Pololu
3. Plano de los servomotores Dynamixel Rx24 y Rx28
4. Documentación de la librería Arduino - Dynamixel
5. Documentación tarjeta MD25
6. Documentación Dynamixel
7. Configuración previa de los servomotores Dynamixel
8. Programas para modificar la dirección de la tarjeta MD25
9. Programas de Arduino
10. Script para generar gráficas

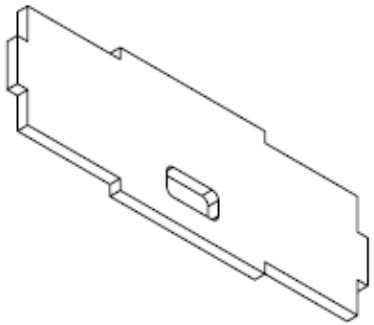
APÉNDICE 1

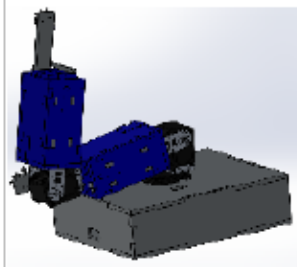
Planos del robot adaptable



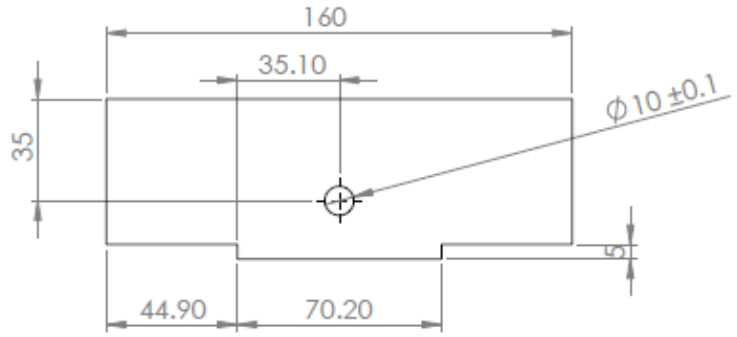


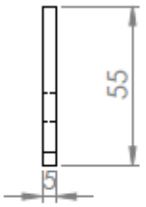


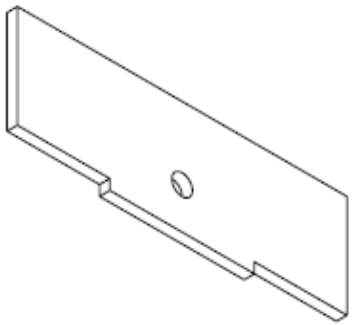


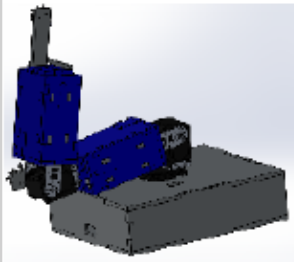


Dibujo N° 1	Escala 1:2	Acotación: mm
Prototipo del robot adaptable	Fecha: 25/02/16	A4
N° Piezas:1	Caja lados corto	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

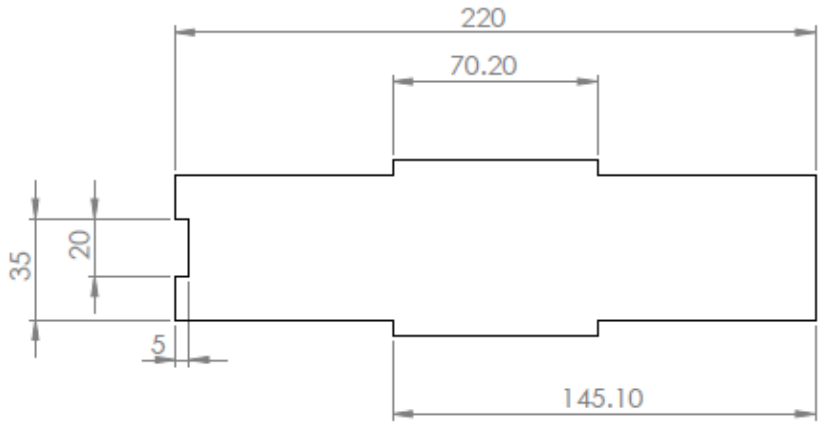


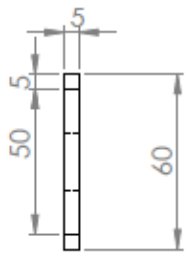


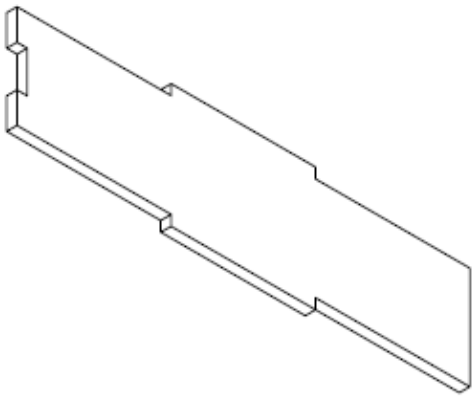


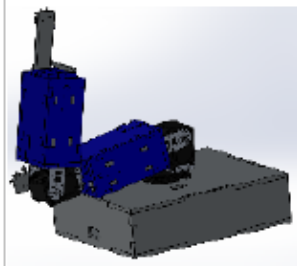


Dibujo N° 2	Escala 1:2	Acotación: mm
Prototipo del robot adaptable	Fecha: 25/02/16	A4
N° Piezas:1	Caja lados corto bisagra bueno	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

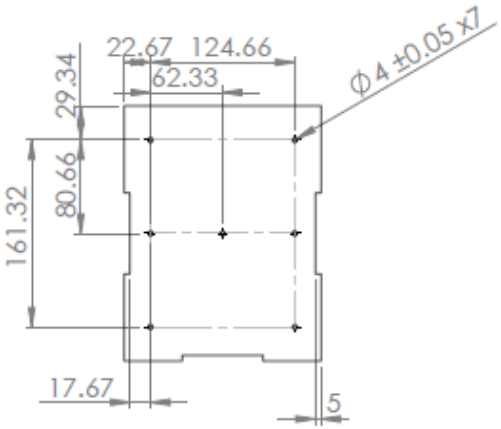


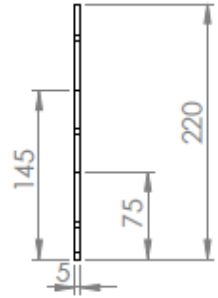


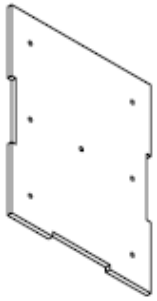


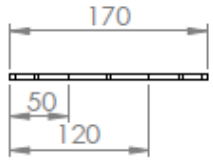


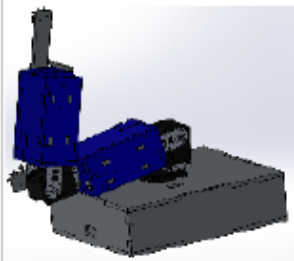
Dibujo N° 3	Escala 1:2	Acotación: mm
Prototipo del robot adaptable	Fecha: 25/02/16	A4
N° Piezas:2	Caja lados largo	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		



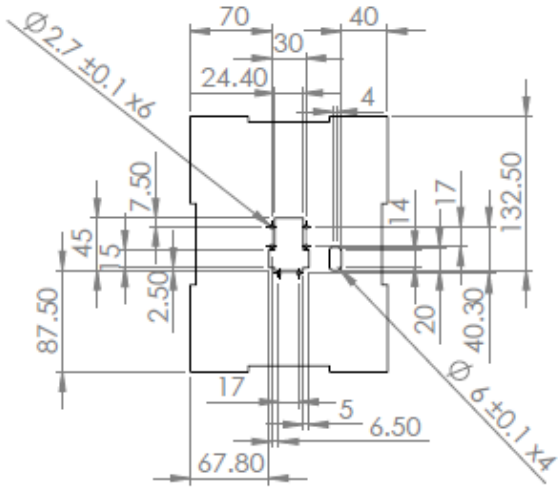


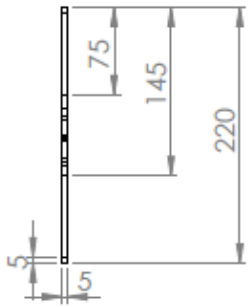


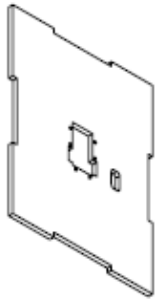


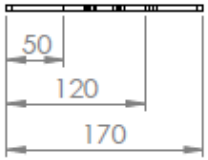


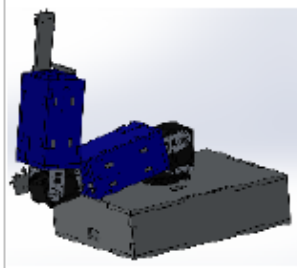
Dibujo N° 4	Escala 1:5	Acotación: mm
Prototipo del robot adaptable	Fecha: 25/02/16	A4
N° Piezas:1	Caja tapa abajo	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		



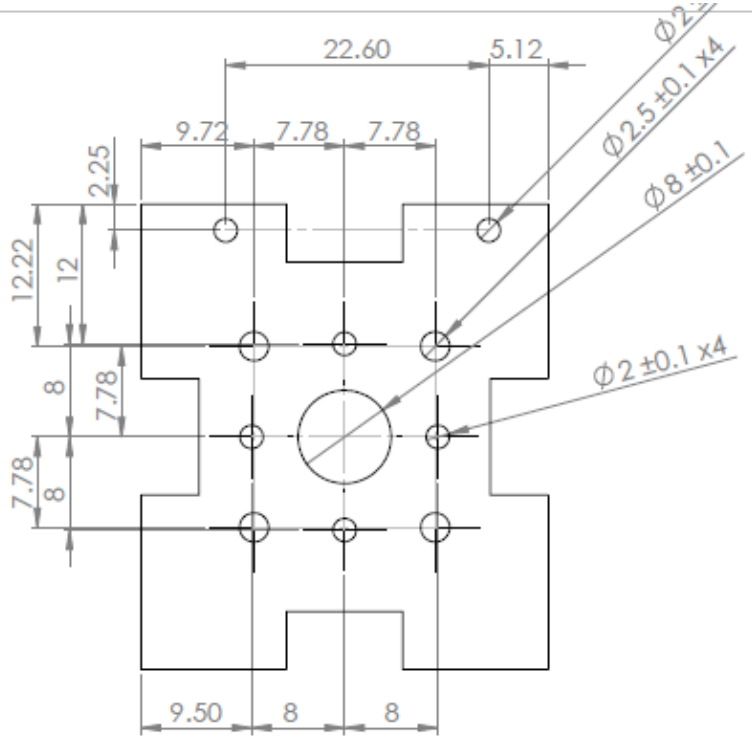


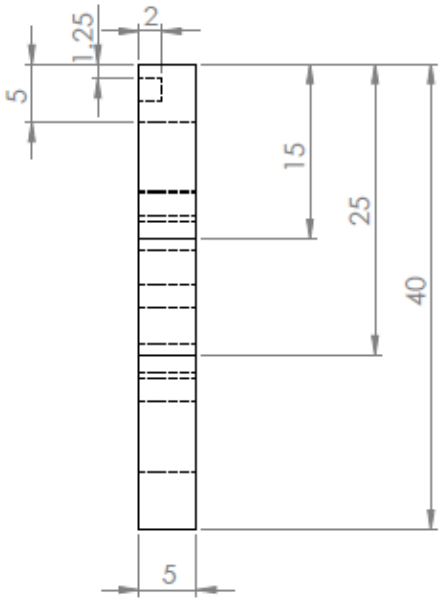


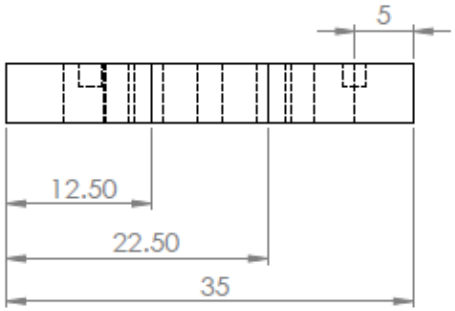


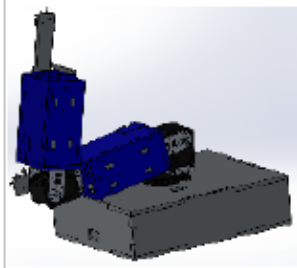


Dibujo N° 5	Escala 1:5	Acotación: mm
Prototipo del robot adaptable	Fecha: 25/02/16	A4
N° Piezas:1	Caja tapa arriba	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

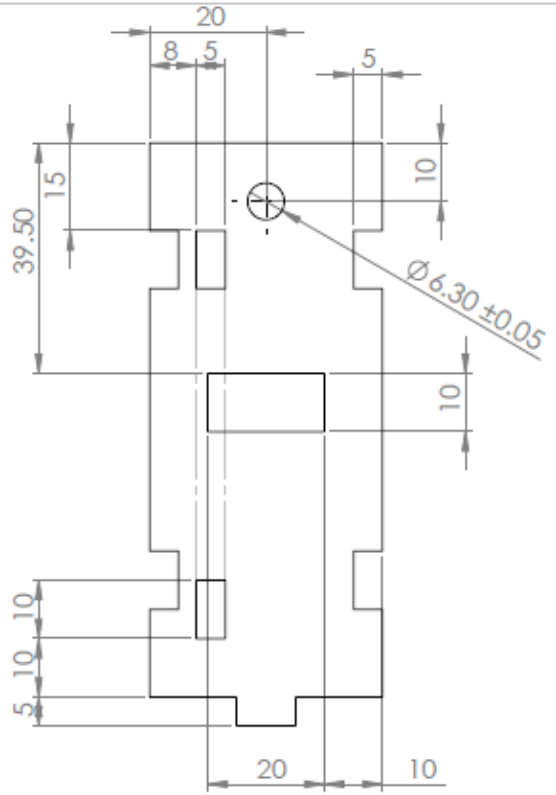


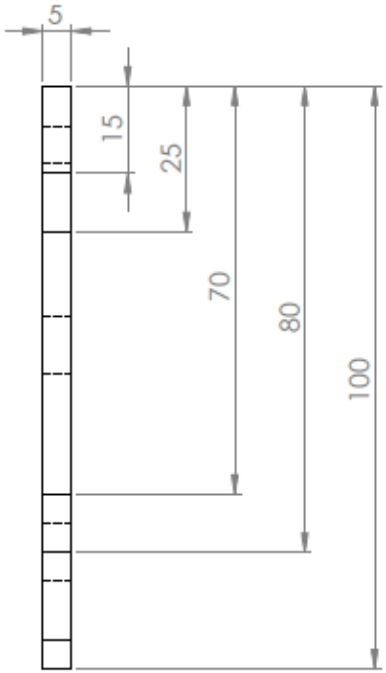


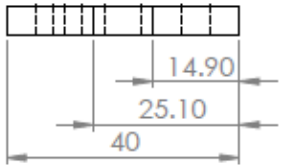


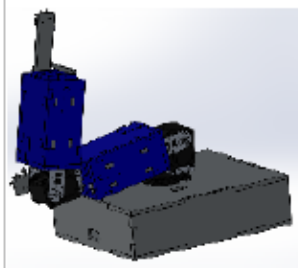


Dibujo N° 6	Escala 2:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 25/02/16	A4
N° Piezas:2	Tapa interior	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

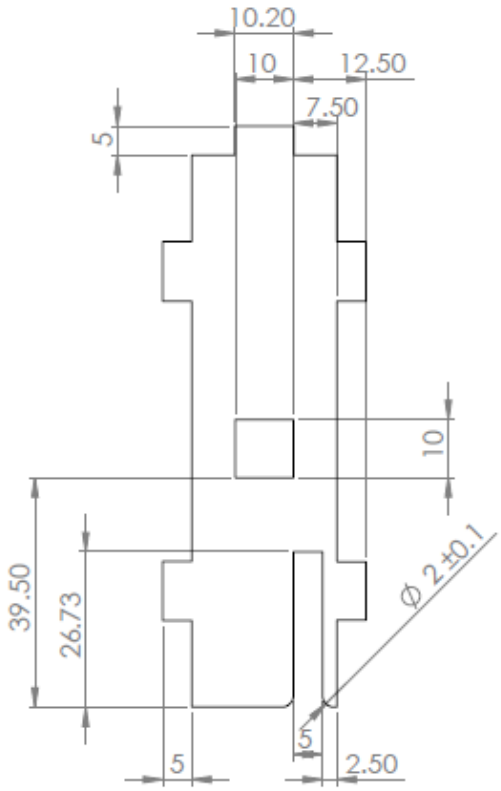


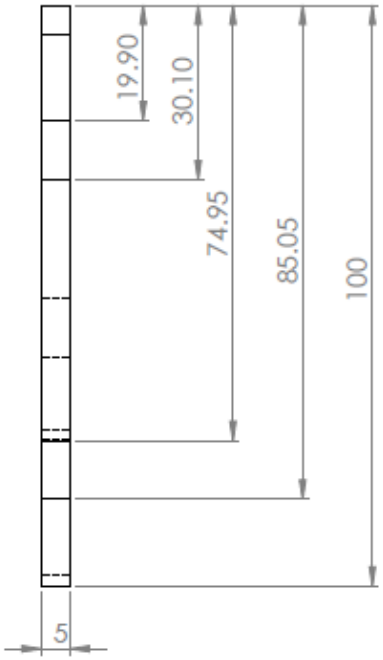


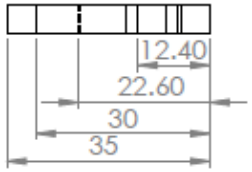


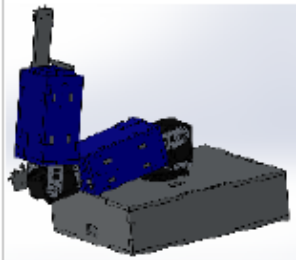


Dibujo N° 7	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 29/02/16	A4
N° Piezas:4	M soporte rodamiento	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

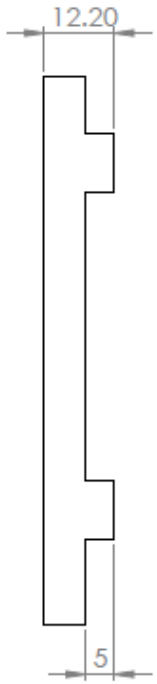


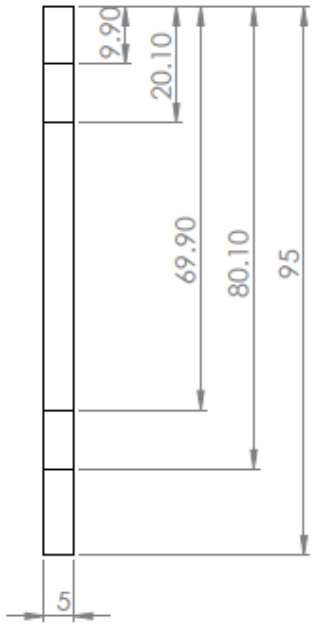


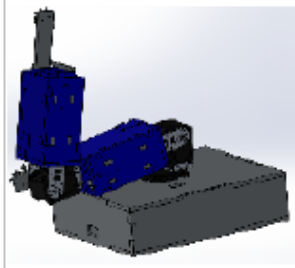




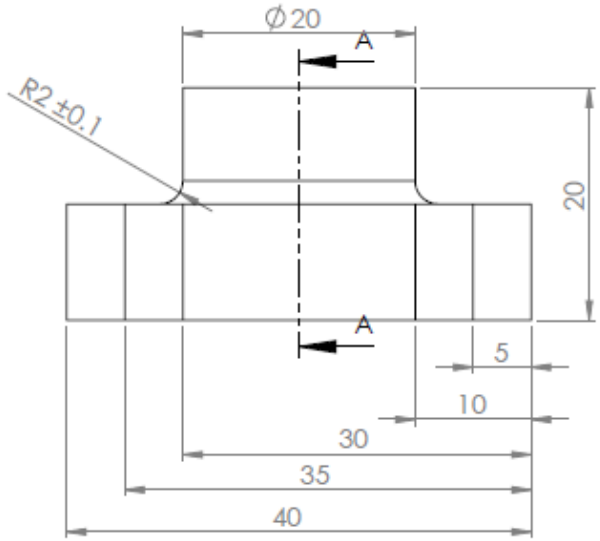
Dibujo N° 8	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 29/02/16	A4
N° Piezas:4	M soporte rodamiento lados	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

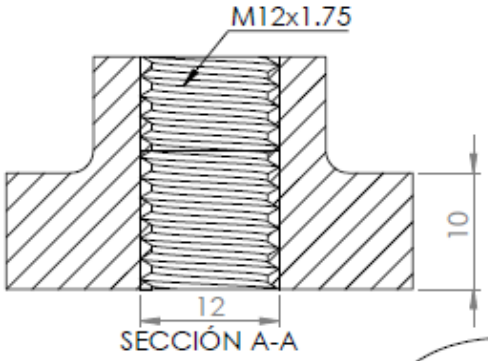


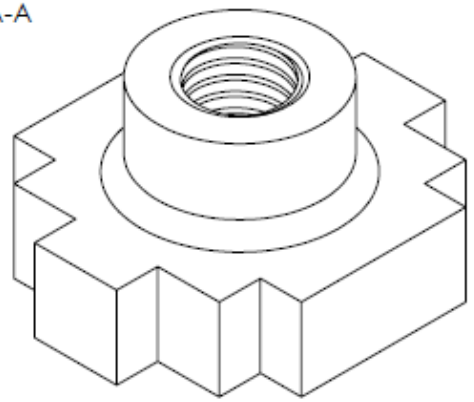


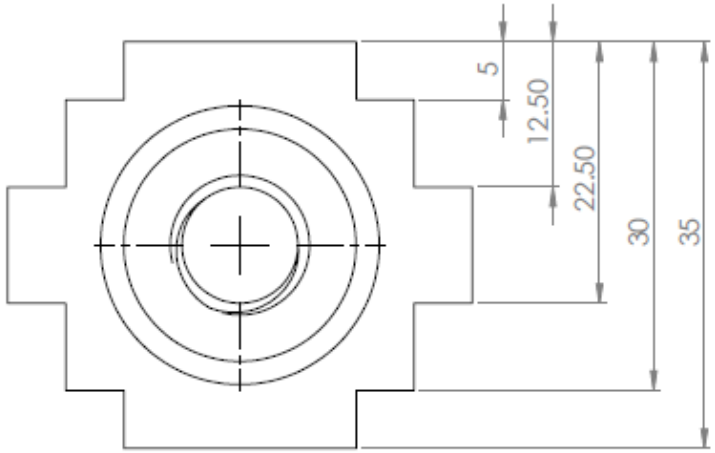


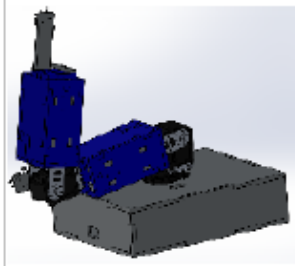
Dibujo N° 9	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 29/02/16	A4
N° Piezas:4	Pestaña rodamiento	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		



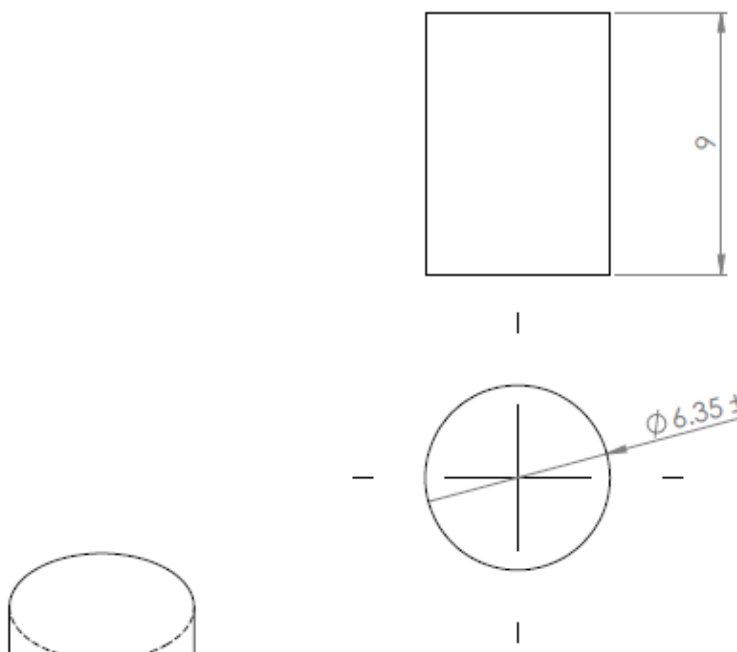


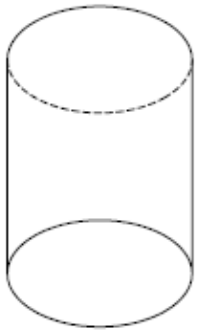


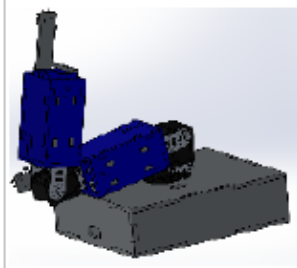




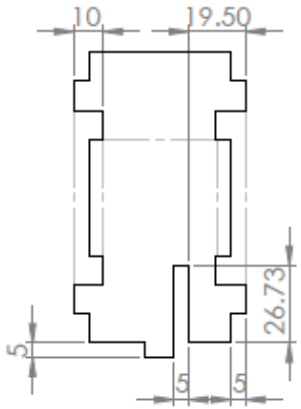
Dibujo Nº 10	Escala 2:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 25/02/16	A4
Nº Piezas:2	Hembra tornillo	
Material:ABS	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

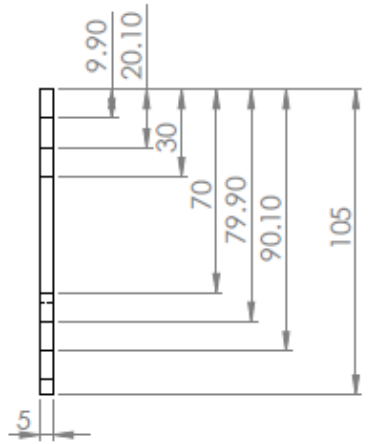


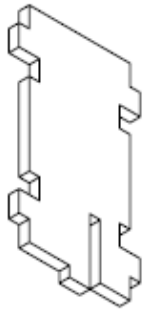


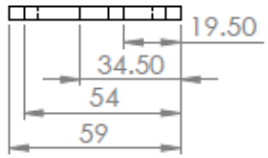


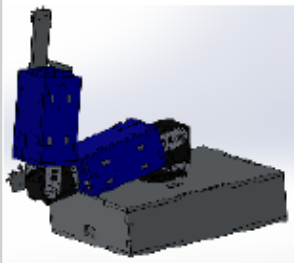
Dibujo N° 11	Escala 5:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 29/02/16	A4
N° Piezas:8	Perno rodamiento interior	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		



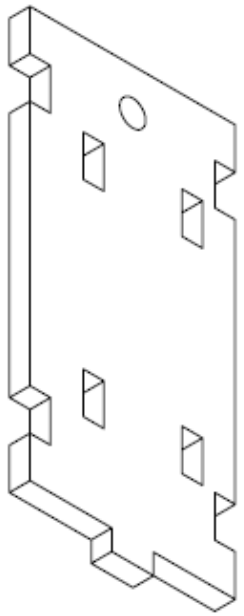


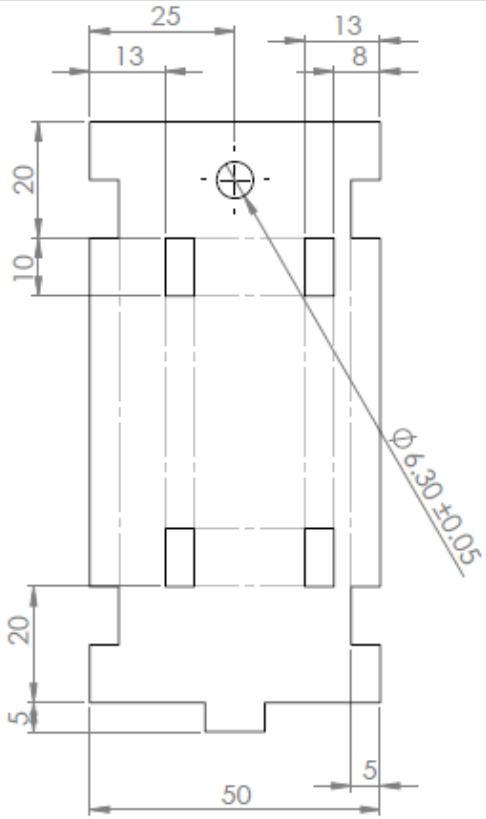


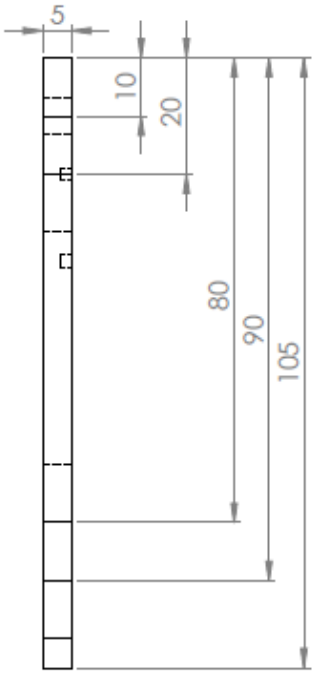


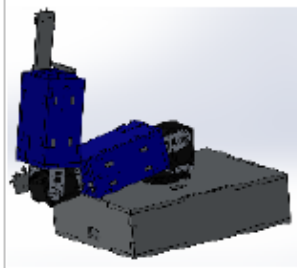


Dibujo N° 12	Escala 1:2	Acotación: mm
Prototipo del robot adaptable	Fecha: 29/02/16	A4
N° Piezas:4	Exterior tapa	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

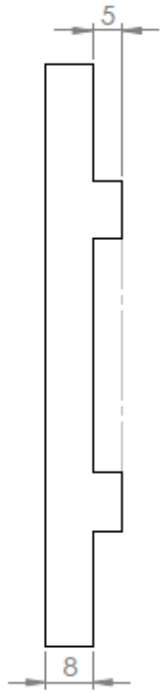


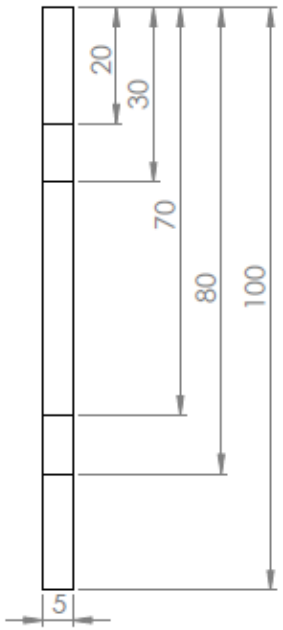


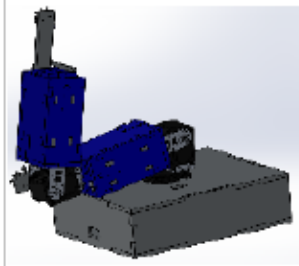




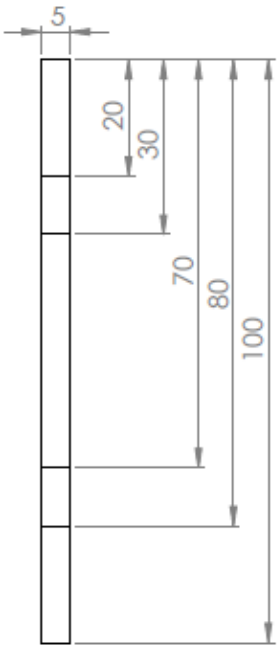
Dibujo N° 13	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 1/03/16	A4
N° Piezas:4	Exterior rod tapa	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

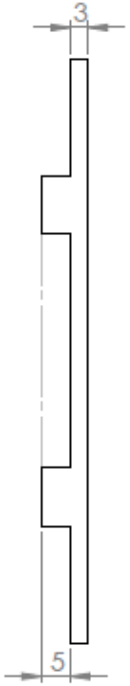


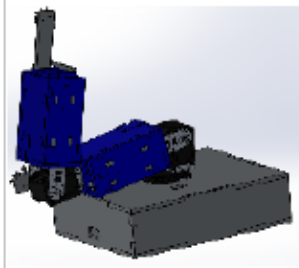




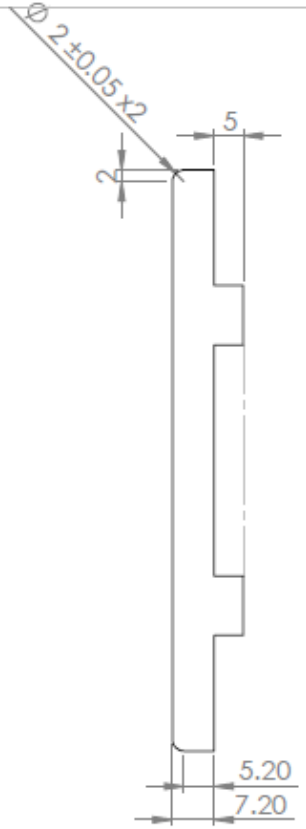
Dibujo N° 14	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 1/03/16	A4
N° Piezas:4	Exterior lados riel	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

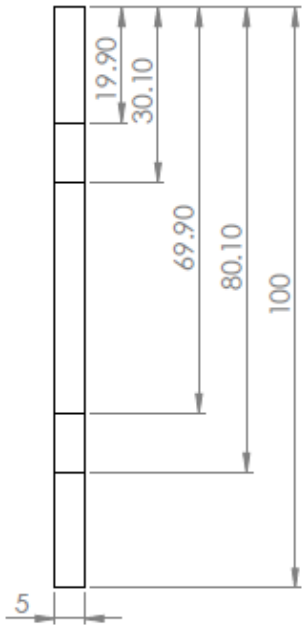


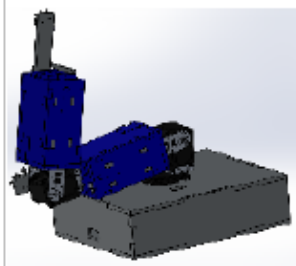




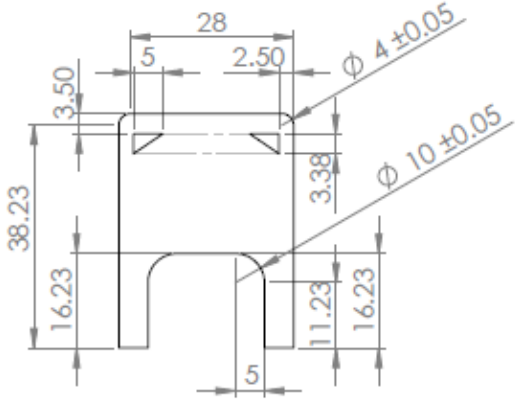
Dibujo N°15	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 1/03/16	A4
N° Piezas:4	Exterior lados riel chico	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

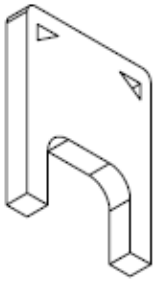


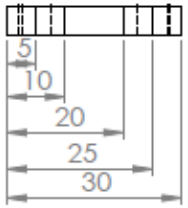


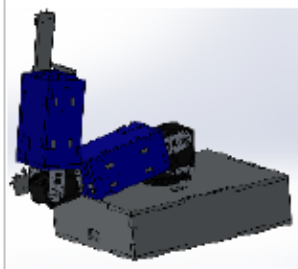


Dibujo N° 16	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 1/03/16	A4
N° Piezas:8	Ayuda soporte interior	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

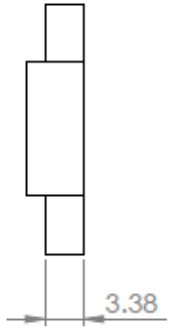


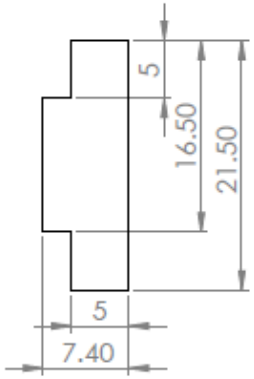


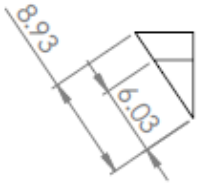


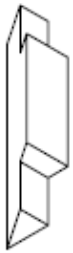


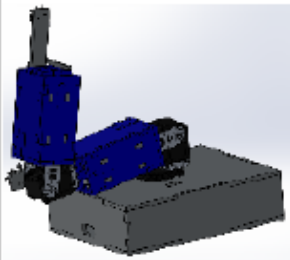
Dibujo Nº 17	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 2/03/16	A4
Nº Piezas:2	Soporte pololu 2	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		



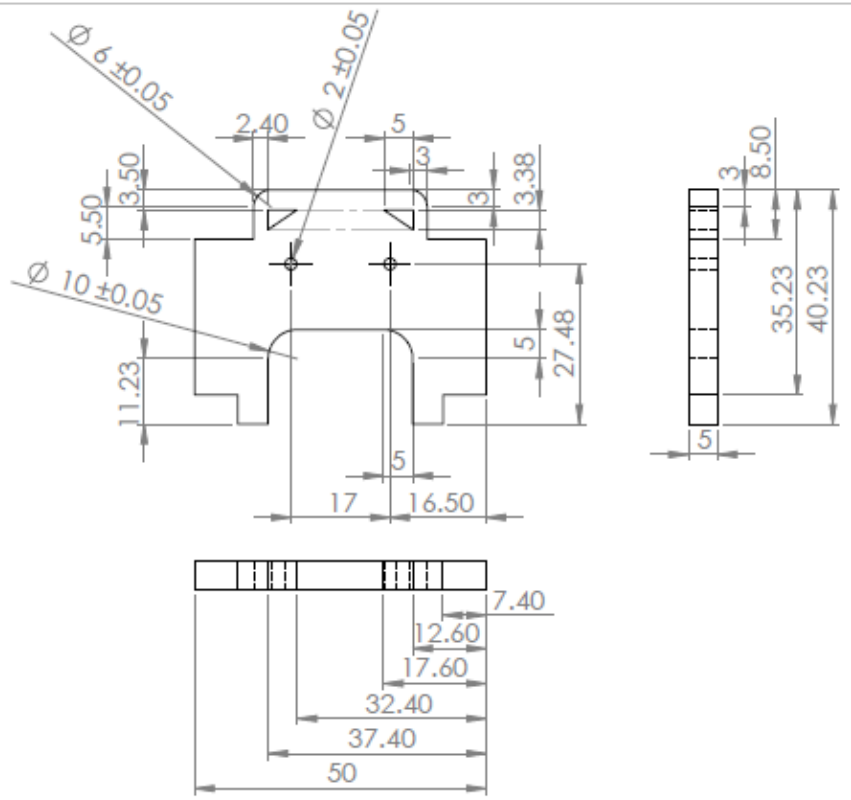


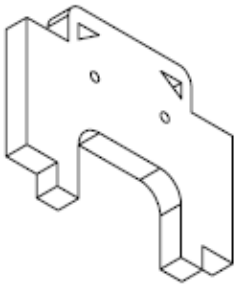


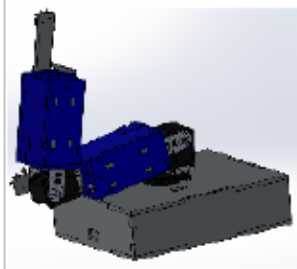




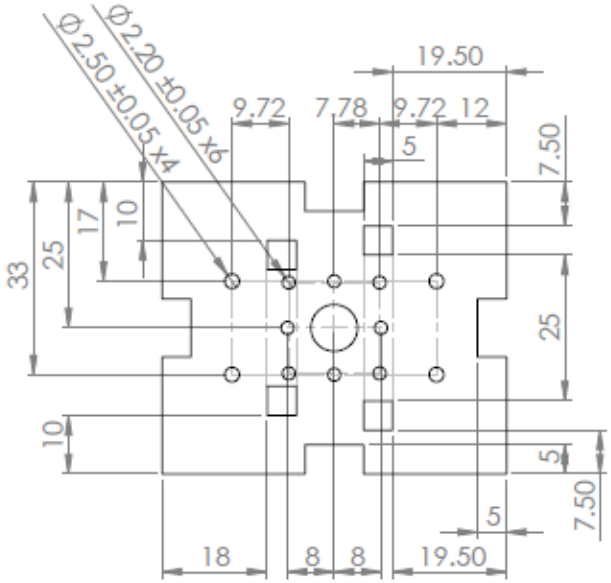
Dibujo N° 18	Escala 2:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 2/03/16	A4
N° Piezas:4	Poste soporte pololu	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

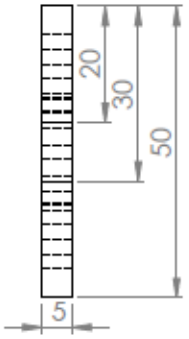


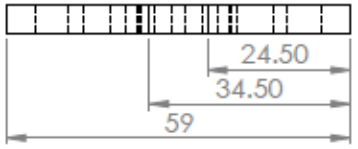


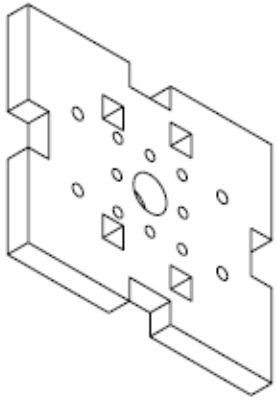


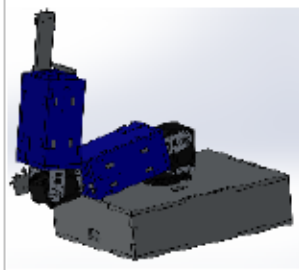
Dibujo N° 19	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 2/03/16	A4
N° Piezas:2	Soporte para base pololu	
Material:Acrílico	Tol:+/- 0.05mm	
Dibujó: Erick E. Pedro Mendoza		



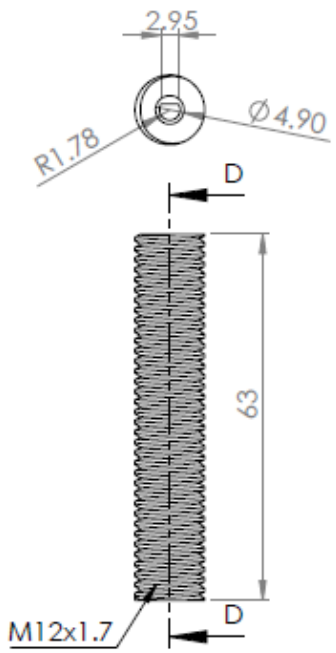


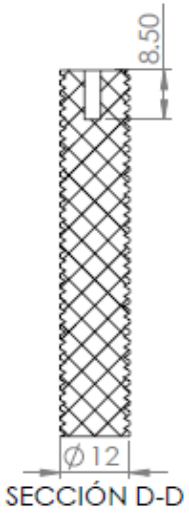




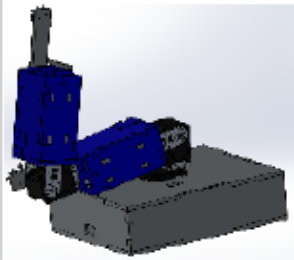


Dibujo N° 20	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 2/03/16	A4
N° Piezas:2	Tapa exterior	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

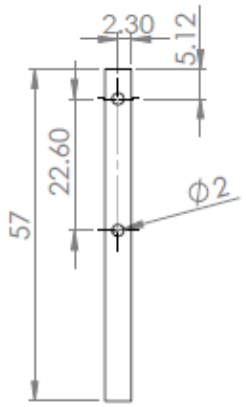


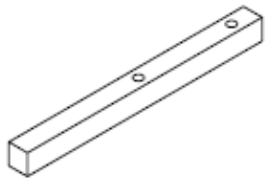


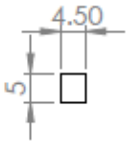
SECCIÓN D-D

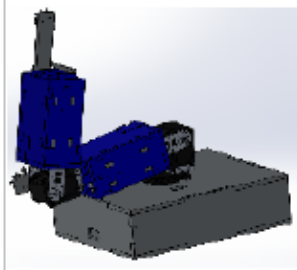


Dibujo N° 21	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 2/03/16	A4
N° Piezas:2	Tornillo ensamble	
Material:ABS	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

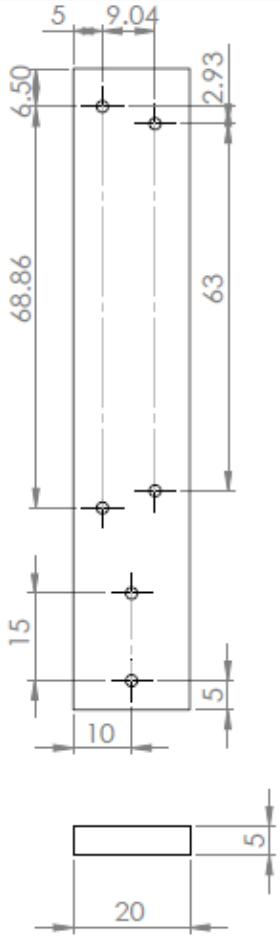


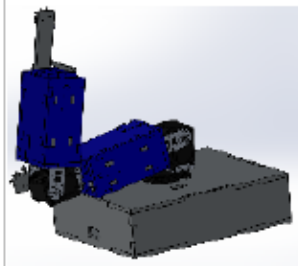






Dibujo N° 22	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 19/04/16	A4
N° Piezas:2	Poste m switch	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		



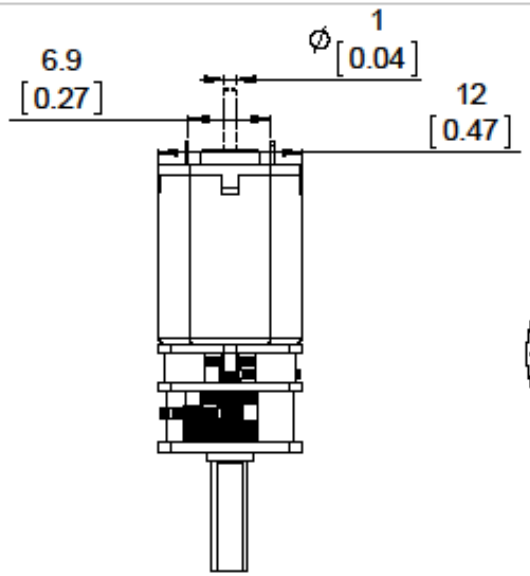


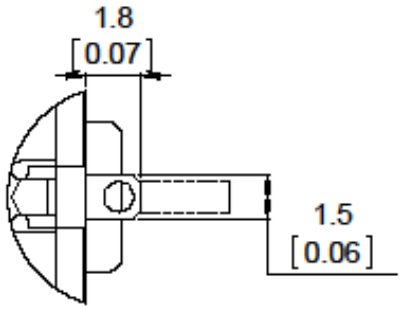
Dibujo N° 23	Escala 1:1	Acotación: mm
Prototipo del robot adaptable	Fecha: 19/04/16	A4
N° Piezas:2	Soporte micro switch	
Material:Acrílico	Tol:+/- 0.1mm	
Dibujó: Erick E. Pedro Mendoza		

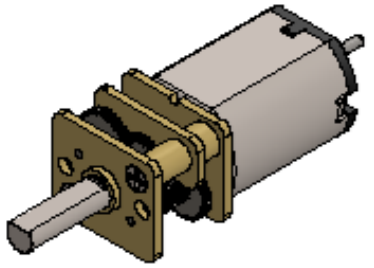
APÉNDICE 2

Plano del Micro-motorreductor Pololu

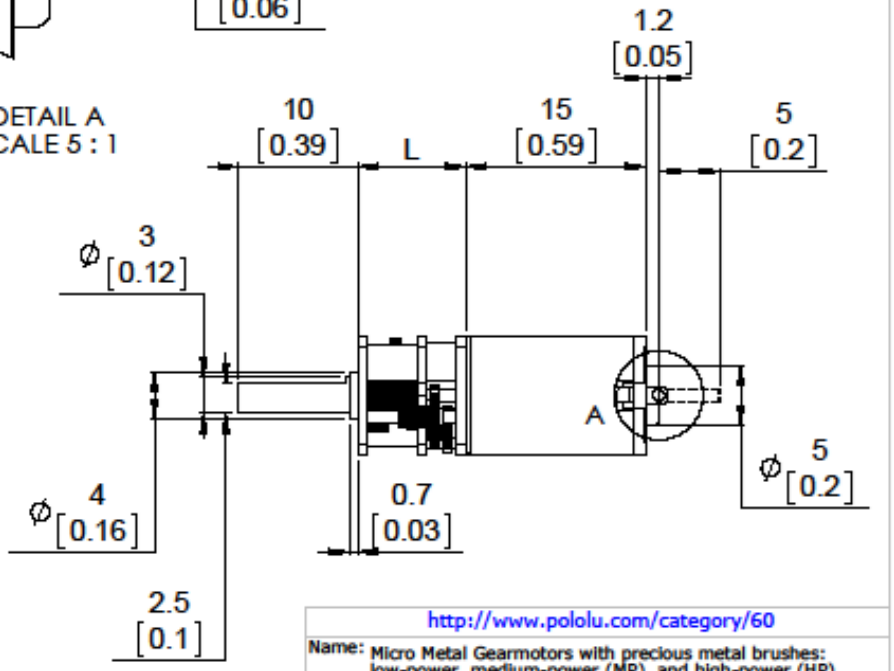
Gear ratio	L
1000:1	12.5 mm [0.49 in]
All others	9 mm [0.35 in]

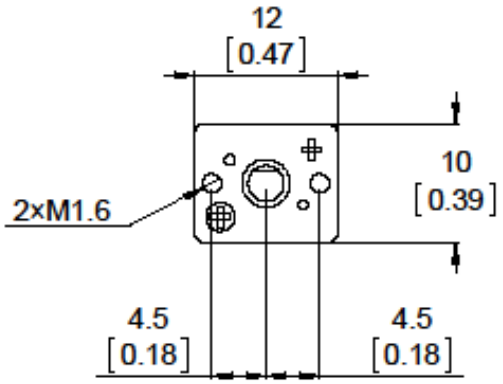






DETAIL A
SCALE 5 : 1





1. To get the specified scale, select 100% in print settings.
2. These dimensions apply to all micro metal gearmotors that are not labeled "HPCB" (i.e. all ones with precious metal brushes).

Scale: 2:1

http://www.pololu.com/category/60

Name: Micro Metal Gearmotors with precious metal brushes: low-power, medium-power (MP), and high-power (HP)

Drawing date: 6 August 2015

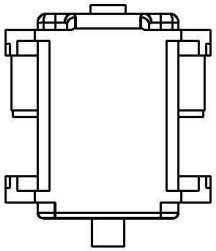
Units: mm [in]

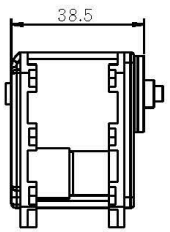
Material: mixed

Pololu Robotics & Electronics

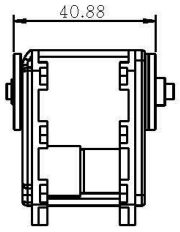
APÉNDICE 3

Plano de los servomotores Dynamixel Rx24 y Rx28

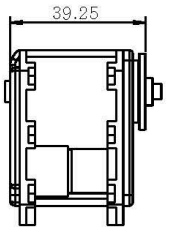




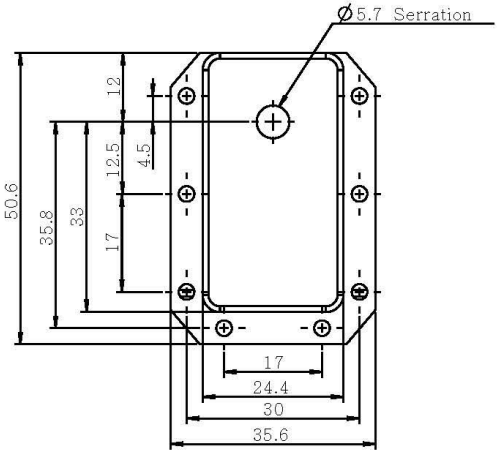
HN07-N101 / HN07-T1
THRUST WASHER
WB M2.5x08

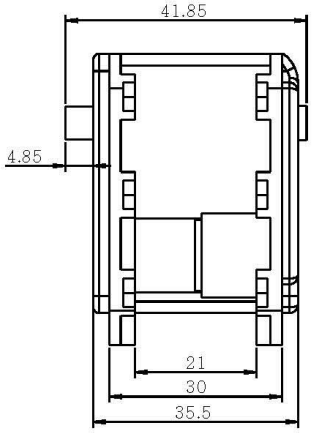


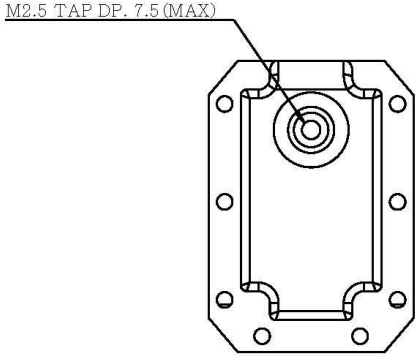
HN07-N101
HN07-i101
THRUST WASHER
BEARING MF106ZZ
RX28 CAP BEARING
WB M2.5x08



HN07-T1
THRUST WASHER
WB M2.5x08





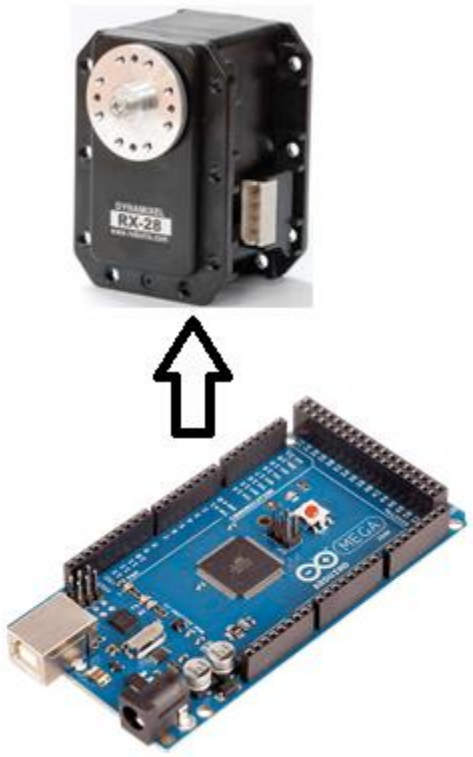


[FOR REFERENCE ONLY]

Title	Material	Scale	Unit	Sheet	ROBOTIS
RX-10 / 24F / 28		NONSCALE	mm	1 of 1	A4
Date 11/07/2011					

APÉNDICE 4

Documentación de la librería Arduino – Dynamixel



SAVAGE ELECTRONICS
Electronics for Everyone

Dynamixel library for Arduino .
Version 1.2.0

begin ()

Description
Initialize the serial communication arduino .

Syntax
begin (baudRate);
begin (baudRate , DATACONTROL)

SoftSerial version
begin (baudRate , rxPin , TxPin)
begin (baudRate , rxPin , TxPin , DATACONTROL)

Parameters
baudRate - serial transmission rate in bps
DATACONTROL - pin control for data transmission and recepcionde
RxPin - pin for receiving data
TxPin - pin for data transmission

Example
Dynamixel.begin (1000000) ; Dynamixel.begin (1000000 2) ;
SoftSerial Version: Dynamixel.begin (1000000 , 2, 3) ;
Dynamixel.begin (1000000 , 2 , 3 , 4) ;

ping ( )

Description
Send a question to the servo motor status .

Syntax
ping (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.ping (1) ;

reset ()

Description
Return to the factory settings of the servomotor.

Syntax
reset (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.reset (1) ;

setId ()

Description
Change the ID of the servomotor.

Syntax
setId (ID , newID);

Parameters
ID - identification number of the servomotor newID - new servomotor ID

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setID (1, 2) ;

setBD ( )

Description
Change the Baud Rate of the servomotor.

Syntax
setBD (ID , baudRate);

Parameters
ID - identification number of the servomotor buadRate - serial transmission speed in bps

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setBD (1, 115200);

move ()

Description
Move the actuator to the position indicated.

Syntax
move (ID , Position) ;

Parameters
ID - identification number of the servomotor
Position - servo position 0 to 1023 (0 to 300 degrees)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.move (1, 512) ;

movespeed ()

Description
Move the actuator to the position indicated airspeed .

Syntax
movespeed (ID , Position, Speed) ;

Parameters
ID - identification number of the servomotor
Position - servo position 0 to 1023 (0 to 300 degrees) Speed - speed that will move the servo 0 to 1023

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.moveSpeed (1, 512 , 1023);

moveRW ( )

Description
Save the instruction that moves the actuator to the position indicated.

Syntax
moveRW (ID , Position) ;

Parameters
ID - identification number of the servomotor
Position - servo position 0 to 1023 (0 to 300 degrees)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.moveRW (1, 512) ;

moveSpeedRW ( )

Description
Save the instruction that moves the actuator to the position indicated airspeed .

Syntax
moveSpeedRW (ID , Position, Speed) ;

Parameters
ID - identification number of the servomotor
Position - servo position 0 to 1023 (0 to 300 degrees)
Speed - speed that will move the servo 0 to 1023

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.moveSpeedRW (1, 512 , 1023);

action ()

Description
Executes the instruction stored in the servomotor.

Syntax
action () ;

Parameters
none

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.action ( ) ;

setEndless ( )

Description
Enables or disables continuous mode servomotor rotation .

Syntax
setEndless (ID , Status);

Parameters
ID - identification number of the servomotor
Status - on or off the Endless (ON or OFF) mode

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setEndless (1, ON) ;

turn ()

Description
Servomotor rotates to the right or left and the speed indicated only if in continuous rotation mode .

Syntax
turn (ID , Side, Speed) ;

Parameters
ID - identification number of the servomotor
Side - direction in which to rotate (RIGTH or LEFT)
Speed - speed that will move the servo 0-1020

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.turn (1, LEFT, 1000) ;

torqueStatus ( )

Description
Enables or disables the torque on the servomotor.

Syntax
torqueStatus (ID , Status);

Parameters
ID - identification number of the servomotor
Status - on or off the touch (ON or OFF)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.torqueStatus (1 , ON) ;

LEDStatus ()

Description
Turns the LED on the back of the servomotor.

Syntax
LEDStatus (ID , Status);

Parameters
ID - identification number of the servomotor
Status - on or off (ON or OFF) LED

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.ledStatus (1, ON) ;

setTempLimit ( )

Description
Configures a maximum operating temperature of the servomotor.

Syntax
setTempLimit (ID , Temperature) ;

Parameters
ID - identification number of the servomotor
Temperature - the maximum temperature to which the servomotor work

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setTempLimit (1, 80) ;

setAngleLimit ( )

Description
Sets a maximum angle CW and CCW operating servomotor.

Syntax
setAngleLimit (ID , CW , CCW);

Parameters
ID - identification number of the servomotor
CW - maximum angle to clockwise
CCW - maximum angle against clockwise

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setAngleLimit (1, 45 , 45) ;

setVoltageLimit ( )

Description
Set a minimum and maximum operating voltage on the actuator .

Syntax
setVoltageLimit (ID , minVoltage , maxVoltage);

Parameters
ID - identification number of the servomotor
minVoltage - minimum operating voltage of the servomotor maxVoltage - maximum operating voltage of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setVoltageLimit (1, 70 , 160) ;

setMaxTorque ( )

Description
Sets a maximum torque on the actuator .

Syntax
setMaxTorque (ID , Maxtorque);

Parameters
ID - identification number
Maxtorque - servomotor maximum torque (0-1023)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setMaxTorque (1, 1023) ;

setSRL ( )

Description
Sets the Status Return Level of servomotor.

Syntax
setSRL (ID , SRL);

Parameters
ID - identification number of the servomotor
SRL - (0 Return none), (read Return 1) , (2 Return all)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setSRL (1, 2) ;

setRDT ( )

Description
Return Delay Time Sets the servomotor .

Syntax
setRDT (ID , RDT);

Parameters
ID - identification number of the servomotor
*RDT - time information return (0-255) * 2us*

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setRDT (1, 255) ;

setLEDAlarm ( )

Description
Set the alarm LED servomotor.

Syntax
setLEDAlarm (ID , LEDAlarm);

Parameters
ID - identification number of the servomotor
LEDAlarm - alarm LED (0-255)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setLEDAlarm (1 , 255) ;

setShutdownAlarm ( )

Description
Set the alarm off the booster.

Syntax
setShutdownAlarm (ID , shutdownAlarm);

Parameters
ID - identification number of the servomotor shutdownAlarm - shutdown

alarm (0-255)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setShutdownAlarm (1, 255) ;

setCMargin ( )

Description
Compliance Margin Sets the servomotor .

Syntax
setCMargin (ID , CWCM , CCWCM);

Parameters
ID - identification number of the servomotor CWCM - CW Compliance Margin (0-255) CCWCM - CCW Compliance Margin (0-255)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setCMargin (1, 1 , 1) ;

setCSlope ( )

Description
Set the servomotor Compliance Slope .

Syntax
setCSlope (ID , CWCS , CCWCS);

Parameters
ID - identification number of the servomotor CWCS - CW Compliance Slope (0-255) CCWCS - CCW Compliance Slope (0-255)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setCSlope (1, 64, 64) ;

setPunch ( )

Description
Punch Sets the maximum current or servomotor.

Syntax
setPunch (ID , Punch);

Parameters
ID - identification number of the servomotor
Punch - current in the servomotor (0-1023)

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.setPunch (1, 1023) ;

moving ()

Description
Check or read if the servomotor is moving.

Syntax
moving (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called
-0 If the actuator is not in motion
-1 If the servo is still moving

Example
var = Dynamixel.moving int (1) ;

RWStatus ( )

Description
Lee REG_WRITE state servomotor .

Syntax
RWStatus (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called
0 if the servo does not have a saved instruction
1 if the actuator has a saved statement

Example
var = Dynamixel.RWStatus int (1) ;

lockRegister ( )

Description
Blocks 24 to 35 records of the servomotor.

Syntax
lockRegister (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called

Example
Dynamixel.lockRegister (1) ;

readTemperature ( )

Description
Reads the internal temperature of the servomotor .

Syntax
readTemperature (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called
 # Internal temperature of the servomotor

Example
var = Dynamixel.readTemperature int (1) ;

readVoltage ( )

Description
Read the supply voltage of the servomotor.

Syntax
readVoltage (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called
 # Supply voltage servomotor

Example
var = Dynamixel.readVoltage int (1) ;

readPosition ( )

Description
Reads the position in which the actuator is located .

Syntax
readPosition (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called
- # Position of the servomotor

Example
var = Dynamixel.readPosition int (1) ;

readSpeed ( )

Description
Read the rpm of the servomotor.

Syntax
readSpeed (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called
- # Speed in rpm of the servomotor

Example
var = Dynamixel.readSpeed int (1) ;

readLoad ( )

Description
Read the current used by the servomotor.

Syntax
readLoad (ID) ;

Parameters
ID - identification number of the servomotor

returns
-1 If there was no response from the servomotor
- # Error found servomotor called
- # Used by current servomotor

Example
var = Dynamixel.readLoad int (1) ;

APÉNDICE 5

Documentación tarjeta MD25

MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive

I2C mode documentation

(Click here for Serial Mode)

Automatic Speed regulation
By using feedback from the encoders the MD25 is able to dynamically increase power as required. If the required speed is not being achieved, the MD25 will increase power to the motors until it reaches the desired rate or the motors reach there maximum output. Speed regulation can be turned off in the command register.

Automatic Motor Timeout
The MD25 will automatically stop the motors if there is no I2C communications within 2 seconds. This is to prevent your robot running wild if the controller fails. The feature can be turned off, if not required. See the command register.

Controlling the MD25
The MD25 is designed to operate in a standard I2C bus system on addresses from 0xB0 to 0xBE (last bit of address is read/write bit, so even numbers only), with its default address being 0xB0. This is easily changed by removing the Address Jumper or in the software see Changing the I2C Bus Address.

I2C mode allows the MD25 to be connected to popular controllers such as the PICAXE, OOPic and BS2p, and a wide range of micro-controllers like PIC's, AVR's, 8051's etc.
I2C communication protocol with the MD25 module is the same as popular EPROM's such as the 24C04. To read one or more of the MD25 registers, first send a start bit, the module address (0XB0 for example) with the read/write bit low, then the register number you wish to read. This is followed by a repeated start and the module address again with the read/write bit high (0XB1 in this example). You are now able to read one or more registers. The MD25 has 17 registers numbered 0 to 16 as follows;

Register	Name	Read/Write	Description
0	Speed1	R/W	Motor1 speed (mode 0,1) or speed (mode 2,3)
1	Speed2/Turn	R/W	Motor2 speed (mode 0,1) or turn (mode 2,3)
2	Enc1a	Read only	Encoder 1 position, 1st byte (highest), capture count when read
3	Enc1b	Read only	Encoder 1 position, 2nd byte
4	Enc1c	Read only	Encoder 1 position, 3rd byte
5	Enc1d	Read only	Encoder 1 position, 4th (lowest byte)
6	Enc2a	Read only	Encoder 2 position, 1st byte (highest), capture count when read
7	Enc2b	Read only	Encoder 2 position, 2nd byte
8	Enc2c	Read only	Encoder 2 position, 3rd byte
9	Enc2d	Read only	Encoder 2 position, 4th byte (lowest byte)
10	Battery volts	Read only	The supply battery voltage
11	Motor 1 current	Read only	The current through motor 1
12	Motor 2 current	Read only	The current through motor 2
13	Software Revision	Read only	Software Revision Number
14	Acceleration rate	R/W	Optional Acceleration register
15	Mode	R/W	Mode of operation (see below)
16	Command	Write only	Used for reset of encoder counts and module address changes

Speed1 Register
Depending on what mode you are in, this register can affect the speed of one motor or both motors. If you are in mode 0 or 1 it will set the speed and direction of motor 1. The larger the number written to this register, the more power is applied to the motor. A mode of 2 or 3 will control the speed and direction of both motors (subject to effect of turn register).

Speed2/Turn Register
When in mode 0 or 1 this register operates the speed and direction of motor 2. When in mode 2 or 3 Speed2 becomes a Turn register, and any value in this register is combined with the contents of Speed1 to steer the device (see below).

Turn mode
Turn mode looks at the speed register to decide if the direction is forward or reverse. Then it applies a subtraction or addition of the turn value on either motor.

so if the direction is forward
motor speed1 = speed - turn
motor speed2 = speed + turn

else the direction is reverse so
motor speed1 = speed + turn
motor speed2 = speed - turn

If the either motor is not able to achieve the required speed for the turn (beyond the maximum output), then the other motor is automatically changed by the program to meet the required difference.

Encoder registers
Each motor has its encoder count stored in an array of four bytes, together the bytes form a signed 32 bit number, the encoder count is captured on a read of the highest byte (registers 2, 6) and the subsequent lower bytes will be held until another read of the highest byte takes place. The count is stored with the highest byte in the lowest numbered register. The registers can be zeroed at any time by writing 32 (0x20) to the command register.

Battery volts
A reading of the voltage of the connected battery is available in this register. It reads as 10 times the voltage (121 for 12.1v).

Motor 1 and 2 current
A guide reading of the average current through the motor is available in this register. It reads approx ten times the number of Amps (25 at 2.5A).

Software Revision number
This register contains the revision number of the software in the modules PIC16F873 controller - currently 1 at the time of writing.

Acceleration Rate
If you require a controlled acceleration period for the attached motors to reach there ultimate speed, the MD25 has a register to provide this. It works by using a value into the acceleration register and incrementing the power by that value. Changing between the current speed of the motors and the new speed (from speed 1 and 2 registers). So if the motors were traveling at full speed in the forward direction (255) and were instructed to move at full speed in reverse (0), there would be 255 steps with an acceleration register value of 1, but 128 for a value of 2. The default acceleration value is 5, meaning the speed is changed from full forward to full reverse in 1.25 seconds. The register will accept values of 1 up to 10 which equates to a period of only 0.65 seconds to travel from full speed in one direction to full speed in the opposite direction.

So to calculate the time (in seconds) for the acceleration to complete :

if new speed > current speed
steps = (new speed - current speed) / acceleration register

if new speed < current speed
steps = (current speed - new speed) / acceleration register

time = steps * 25ms

For example :

Acceleration register	Time/step	Current speed	New speed	Steps	Acceleration time
1	25ms	0	255	255	6.375s
2	25ms	127	255	64	1.6s
3	25ms	80	0	27	0.675s
5 (default)	25ms	0	255	51	1.275s

10	25ms	255	0	26	0.65s

Mode Register
The mode register selects which mode of operation and I2C data input type the user requires. The options being:
0, (Default Setting) If a value of 0 is written to the mode register then the meaning of the speed registers is literal speeds in the range of 0 (Full Reverse) 128 (Stop) 255 (Full Forward).

1, Mode 1 is similar to Mode 0, except that the speed registers are interpreted as signed values. The meaning of the speed registers is literal speeds in the range of -128 (Full Reverse) 0 (Stop) 127 (Full Forward).

2, Writing a value of 2 to the mode register will make speed1 control both motors speed, and speed2 becomes the turn value.
Data is in the range of 0 (Full Reverse) 128 (Stop) 255 (Full Forward).

3, Mode 3 is similar to Mode 2, except that the speed registers are interpreted as signed values.
Data is in the range of -128 (Full Reverse) 0 (Stop) 127 (Full Forward)

Command register

Command		Action
Dec	Hex	
32	20	Resets the encoder registers to zero
48	30	Disables automatic speed regulation
49	31	Enables automatic speed regulation (default)
50	32	Disables 2 second timeout of motors (Version 2 onwards only)
51	33	Enables 2 second timeout of motors when no I2C comms (default) (Version 2 onwards only)
160	A0	1st in sequence to change I2C address
170	AA	2nd in sequence to change I2C address
165	A5	3rd in sequence to change I2C address

Changing the I2C Bus Address
To change the I2C address of the MD25 by writing a new address you must have only one module on the bus. Write the 3 sequence commands in the correct order followed by the address. Example; to change the address of an MD25 currently at 0xB0 (the default shipped address) to 0xB4, write the following to address 0xB0; (0xA0, 0xAA, 0xA5, 0xB4). These commands must be sent in the correct sequence to change the I2C address, additionally, no other command may be issued in the middle of the sequence. The sequence must be sent to the command register at location 16, which means 4 separate write transactions on the I2C bus. Because of the way the MD25 works internally, there MUST be a delay of at least 5mS between the writing of each of these 4 transactions. When done, you should label the MD25 with its address, however if you do forget, just power it up without sending any commands. The MD25 will flash its address out on the green communication LED. One long flash followed by a number of shorter flashes indicating its address. Any command sent to the MD25 during this period will still be received and writing new speeds or a write to the command register will terminate the flashing.

Address		Long Flash	Short Flashes
Decimal	Hex		
176	B0	1	0
178	B2	1	1
180	B4	1	2
182	B6	1	3
184	B8	1	4
186	BA	1	5
188	BC	1	6
190	BE	1	7

Take care not to set more than one MD25 to the same address, there will be a bus collision and very unpredictable results.

APÉNDICE 6

Documentación Dynamixel

Show

ROBOTIS e-Manual v1.25.00

RX-28

Parts Photo

[RX-28]

H/W Specification

- Weight : 72g
- Dimension : 35.6mm x 50.6mm x 35.5mm
- Resolution : 0.29 ;
- Gear Reduction Ratio : 193 : 1
- Stall Torque : 3.7N.m (at 18.5V, 1.9A)
- No load speed : 85rpm (at 18.5V)
- Running Degree
  - 0 ~ 300 ; ;
  - Endless Turn
- Running Temperature : -5° ~ +80°
- Voltage : 12V~18.5V (Recommended Voltage 14.8V)
- Command Signal : Digital Packet
- Protocol Type : RS485 Asynchronous Serial Communication (8bit,1stop, No Parity)
- Link (Physical) : RS485 Multi Drop Bus
- ID : 254 ID (0~253)
- Communication Speed : 7343bps ~ 1 Mbps
- Feedback : Position, Temperature, Load, Input Voltage, etc.
- Material : Full Metal Gear, Engineering Plastic Body
- Standby current : 50 mA

Stall torque is the maximum instantaneous and static torque
Stable motions are possible with robots designed for loads with 1/5 or less of the stall torque

Control Table

Control Table consists of data regarding the current status and operation, which exists inside of Dynamixel. The user can control Dynamixel by changing data of Control Table via Instruction Packet.

EEPROM and RAM

Data in RAM area is reset to the initial value whenever the power is turned on while data in EEPROM area is kept once the value is set even if the power is turned off.

Address

It represents the location of data. To read from or write data to Control Table, the user should assign the correct address in the Instruction Packet.

Access

Dynamixel has two kinds of data: Read-only data, which is mainly used for sensing, and Read-and-Write data, which is used for driving.

Initial Value

In case of data in the EEPROM Area, the initial values on the right side of the below Control Table are the factory default settings. In case of data in the RAM Area, the initial values on the right side of the above Control Tables are the ones when the power is turned on.

Highest/Lowest Byte

In the Control table, some data share the same name, but they are attached with (L) or (H) at the end of each name to distinguish the address. This data requires 16bit, but it is divided into 8bit each for the addresses (low) and (high). These two addresses should be written with one Instruction Packet at the same time.

Area	Address (Hexadecimal)	Name	Description	Access	Initial Value (Hexadecimal)
EEPROM	0 (0X00)	Model Number(L)	Lowest byte of model number	R	28 (0X1C)
	1 (0X01)	Model Number(H)	Highest byte of model number	R	0 (0X00)
	2 (0X02)	Version of Firmware	Information on the version of firmware	R	-
	3 (0X03)	ID	ID of Dynamixel	RW	1 (0X01)
	4 (0X04)	Baud Rate	Baud Rate of Dynamixel	RW	34 (0X22)
	5 (0X05)	Return Delay Time	Return Delay Time	RW	250 (0XFA)
	6 (0X06)	CW Angle Limit(L)	Lowest byte of clockwise Angle Limit	RW	0 (0X00)
	7 (0X07)	CW Angle Limit(H)	Highest byte of clockwise Angle Limit	RW	0 (0X00)
	8 (0X08)	CCW Angle Limit(L)	Lowest byte of counterclockwise Angle Limit	RW	255 (0XFF)
	9 (0X09)	CCW Angle Limit(H)	Highest byte of counterclockwise Angle Limit	RW	3 (0X03)
	11 (0X0B)	the Highest Limit Temperature	Internal Limit Temperature	RW	80 (0X50)
	12 (0X0C)	the Lowest Limit Voltage	Lowest Limit Voltage	RW	60 (0X3C)
	13 (0X0D)	the Highest Limit Voltage	Highest Limit Voltage	RW	190 (0XBE)
	14 (0X0E)	Max Torque(L)	Lowest byte of Max. Torque	RW	255 (0XFF)
	15 (0X0F)	Max Torque(H)	Highest byte of Max. Torque	RW	3 (0X03)
	16 (0X10)	Status Return Level	Status Return Level	RW	2 (0X02)
	17 (0X11)	Alarm LED	LED for Alarm	RW	36 (0X24)
	18 (0X12)	Alarm Shutdown	Shutdown for Alarm	RW	36 (0X24)
RAM	24 (0X18)	Torque Enable	Torque On/Off	RW	0 (0X00)
	25 (0X19)	LED	LED On/Off	RW	0 (0X00)
	26 (0X1A)	CW Compliance Margin	CW Compliance margin	RW	1 (0X01)
	27 (0X1B)	CCW Compliance Margin	CCW Compliance margin	RW	1 (0X01)
	28 (0X1C)	CW Compliance Slope	CW Compliance slope	RW	32 (0X20)
	29 (0X1D)	CCW Compliance Slope	CCW Compliance slope	RW	32 (0X20)
	30 (0X1E)	Goal Position(L)	Lowest byte of Goal Position	RW	-
	31 (0X1F)	Goal Position(H)	Highest byte of Goal Position	RW	-
	32 (0X20)	Moving Speed(L)	Lowest byte of Moving Speed (Moving Velocity)	RW	-
	33 (0X21)	Moving Speed(H)	Highest byte of Moving Speed (Moving Velocity)	RW	-
	34 (0X22)	Torque Limit(L)	Lowest byte of Torque Limit (Goal Torque)	RW	ADD14
	35 (0X23)	Torque Limit(H)	Highest byte of Torque Limit (Goal Torque)	RW	ADD15
	36 (0X24)	Present Position(L)	Lowest byte of Current Position (Present Velocity)	R	-
	37 (0X25)	Present Position(H)	Highest byte of Current Position (Present Velocity)	R	-
	38 (0X26)	Present Speed(L)	Lowest byte of Current Speed	R	-
	39 (0X27)	Present Speed(H)	Highest byte of Current Speed	R	-
	40 (0X28)	Present Load(L)	Lowest byte of Current Load	R	-
	41 (0X29)	Present Load(H)	Highest byte of Current Load	R	-
	42 (0X2A)	Present Voltage	Current Voltage	R	-
	43 (0X2B)	Present Temperature	Current Temperature	R	-
	44 (0X2C)	Registered	Means if Instruction is registered	R	0 (0X00)
	46 (0X2E)	Moving	Means if there is any movement	R	0 (0X00)
	47 (0X2F)	Lock	Locking EEPROM	RW	0 (0X00)
	48 (0X30)	Punch(L)	Lowest byte of Punch	RW	32 (0X20)
	49 (0X31)	Punch(H)	Highest byte of Punch	RW	0 (0X00)

Address Function Help

EEPROM Area

Model Number

It represents the Model Number.

Firmware Version

It represents the firmware version.

ID

It is a unique number to identify Dynamixel.

The range from 0 to 252 (0xFC) can be used, and, especially, 254(0xFE) is used as the Broadcast ID.

If the Broadcast ID is used to transmit Instruction Packet, we can command to all Dynamixels.

> Please be careful not to duplicate the ID of connected Dynamixel.

Baud Rate

It represents the communication speed. 0 to 254 (0xFE) can be used for it.

This speed is calculated by using the below formula.

Speed(BPS) = 2000000/(Data+1)

Data	Set BPS	Target BPS	Tolerance
1	1000000.0	1000000.0	0.000 %
3	500000.0	500000.0	0.000 %
4	400000.0	400000.0	0.000 %
7	250000.0	250000.0	0.000 %
9	200000.0	200000.0	0.000 %
16	117647.1	115200.0	-2.124 %
34	57142.9	57600.0	0.794 %
103	19230.8	19200.0	-0.160 %
207	9615.4	9600.0	-0.160 %

> Note : Maximum Baud Rate error of 3% is within the tolerance of UART communication.

Return Delay Time

It is the delay time per data value that takes from the transmission of Instruction Packet until the return of Status Packet.

0 to 254 (0xFE) can be used, and the delay time per data value is 2 usec.

That is to say, if the data value is 10, 20 usec is delayed. The initial value is 250 (0xFA) (i.e., 0.5 msec).

CW/CCW Angle Limit

The angle limit allows the motion to be restrained.

The range and the unit of the value is the same as Goal Position(Address 30, 31).

- CW Angle Limit: the minimum value of Goal Position(Address 30, 31)
- CCW Angle Limit: the maximum value of Goal Position(Address 30, 31)

The following two modes can be set pursuant to the value of CW and CCW.

Operation Type	CW / CCW
Wheel Mode	both are 0
Joint Mode	neither at 0

The wheel mode can be used to wheel-type operation robots since motors of the robots spin infinitely.

The joint mode can be used to multi-joints robot since the robots can be controlled with specific angles.

The Highest Limit Temperature

> Caution : Do not set the temperature lower/higher than the default value.
> When the temperature alarm shutdown occurs, wait 20 minutes to cool the temperature before re-use.

Using the product when the temperature is high may and can cause damage.

The Lowest (Highest) Limit Voltage

It is the operation range of voltage.

50 to 250 (0x32 ~ 0x96) can be used. The unit is 0.1V.

For example, if the value is 80, it is 8V.

If Present Voltage (Address42) is out of the range, Voltage Range Error Bit (Bit0) of Status Packet is returned as '1' and Alarm is triggered as set in the addresses 17 and 18.

Max Torque

It is the torque value of maximum output. 0 to 1023 (0x3FF) can be used, and the unit is about 0.1%.

For example, Data 1023 (0x3FF) means that Dynamixel will use 100% of the maximum torque it can produce while Data 512 (0x200) means that Dynamixel will use 50% of the maximum torque. When the power is turned on, Torque Limit (Addresses 34 and 35) uses the value as the initial value.

Status Return Level

It decides how to return Status Packet. There are three ways like the below table.

Value	Return of Status Packet
0	No return against all commands (Except PING Command)
1	Return only for the READ command
2	Return for all commands

When Instruction Packet is Broadcast ID, Status Packet is not returned regardless of Status Return Level.

Alarm LED

Alarm Shutdown

Dynamixel can protect itself by detecting errors occur during the operation.

The errors can be set are as the table below.

Bit	Name	Contents
Bit 7	0	-
Bit 6	Instruction Error	When undefined Instruction is transmitted or the Action command is delivered without the reg_write command
Bit 5	Overload Error	When the current load cannot be controlled with the set maximum torque
Bit 4	CheckSum Error	When the Checksum of the transmitted Instruction Packet is invalid
Bit 3	Range Error	When the command is given beyond the range of usage
Bit 2	OverHeating Error	When the internal temperature is out of the range of operating temperature set in the Control Table
Bit 1	Angle Limit Error	When Goal Position is written with the value that is not between CW Angle Limit and CCW Angle Limit
Bit 0	Input Voltage Error	When the applied voltage is out of the range of operating voltage set in the Control Table

It is possible to make duplicate set since the function of each bit is run by the logic of 'OR'. That is, if 0X05 (binary 00000101) is set, both Input Voltage Error and Overheating Error can be detected.

If errors occur, in case of Alarm LED, the LED blinks; in case of Alarm Shutdown, the motor output becomes 0 % by making the value of Torque Limit(Address 34, 35) as 0.

RAM Area

Torque Enable

Value	Meaning
0	Keeps Torque from generating by interrupting the power of motor.
1	Generates Torque by impressing the power to the motor.

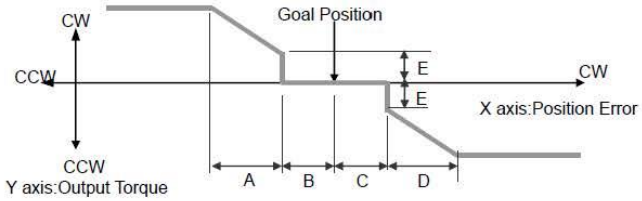
LED

Bit	Meaning
0	Turn OFF the LED
1	Turn ON the LED

Compliance

Compliance is to set the control flexibility of the motor.

The following diagram shows the relationship between output torque and position of the motor.



A : CCW Compliance Slope(Address0x1D)
B : CCW Compliance Margin(Address0x1B)
C : CW Compliance Margin(Address0x1A)
D : CW Compliance Slope (Address0x1C)
E : Punch(Address0x30,31)

Compliance Margin

It exists in each direction of CW/CCW and means the error between goal position and present position.

The range of the value is 0~255, and the unit is the same as Goal Position.(Address 30,31)

The greater the value, the more difference occurs.

Compliance Slope

It exists in each direction of CW/CCW and sets the level of Torque near the goal position.

Compliance Slope is set in 7 steps, the higher the value, the more flexibility is obtained.

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

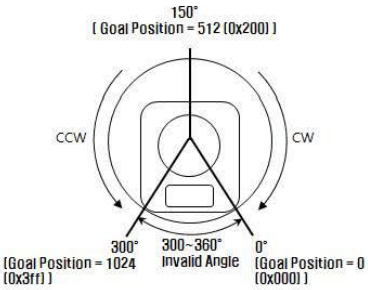
Step	Data Value	Data Representative Value
1	0 (0x00) ~ 3(0x03)	2 (0x02)
2	4(0x04) ~ 7(0x07)	4 (0x04)
3	8(0x08)~15(0x0F)	8 (0x08)
4	16(0x10)~31(0x1F)	16 (0x10)
5	32(0x20)~63(0x3F)	32 (0x20)
6	64(0x40)~127(0x7F)	64 (0x40)
7	128(0x80)~254(0xFE)	128 (0x80)

Goal Position

It is a position value of destination.

0 to 1023 (0x3FF) is available. The unit is 0.29 degree.

If Goal Position is out of the range, Angle Limit Error Bit (Bit1) of Status Packet is returned as '1' and Alarm is triggered as set in Alarm LED/Shutdown.



<The picture above is based on the front of relevant model>

If it is set to Wheel Mode, this value is not used.

Moving Speed
It is a moving speed to Goal Position.
The range and the unit of the value may vary depending on the operation mode.

- Join Mode
  0~1023 (0X3FF) can be used, and the unit is about 0.111rpm.
  If it is set to 0, it means the maximum rpm of the motor is used without controlling the speed.
  If it is 1023, it is about 114rpm.
  For example, if it is set to 300, it is about 33.3 rpm.

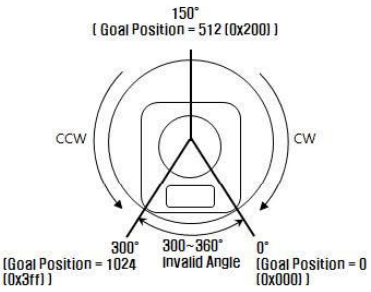
Notes: Please check the maximum rpm of relevant model in Joint Mode. Even if the motor is set to more than maximum rpm, it cannot generate the torque more than the maximum rpm.

- Wheel Mode
  0~2047(0X7FF) can be used, the unit is about 0.1%.
  If a value in the range of 0~1023 is used, it is stopped by setting to 0 while rotating to CCW direction.
  If a value in the range of 1024~2047 is used, it is stopped by setting to 1024 while rotating to CW direction.
  That is, the 10th bit becomes the direction bit to control the direction.
  In Wheel Mode, only the output control is possible, not speed.
  For example, if it is set to 512, it means the output is controlled by 50% of the maximum output.

Torque Limit
It is the value of the maximum torque limit.
0 to 1023 (0x3FF) is available, and the unit is about 0.1%.
For example, if the value is 512, it is about 50%; that means only 50% of the maximum torque will be used.
If the power is turned on, the value of Max Torque (Address 14, 15) is used as the initial value.

Notes: If the function of Alarm Shutdown is triggered, the motor loses its torque because the value becomes 0. At this moment, if the value is changed to the value other than 0, the motor can be used again.

Present Position
It is the current position value of Dynamixel.
The range of the value is 0~1023 (0x3FF), and the unit is 0.29 degree.



<The picture above is based on the front of relevant model>

Caution: If it is set to Wheel Mode, the value cannot be used to measure the moving distance and the rotation frequency.

Present Speed
It is the current moving speed.

0~2047 (0X7FF) can be used.
If a value is in the rage of 0~1023, it means that the motor rotates to the CCW direction.
If a value is in the rage of 1024~2047, it means that the motor rotates to the CW direction.
That is, the 10th bit becomes the direction bit to control the direction, and 0 and 1024 are equal.
The unit of this value varies depending on operation mode.

- Joint Mode
  The unit is about 0.111rpm.
  For example, if it is set to 300, it means that the motor is moving to the CCW direction at a rate of about 33.3rpm.
- Wheel Mode
  The unit is about 0.1%.
  For example, if it is set to 512, it means that the torque is controlled by 50% of the maximum torque to the CCW direction.

Present Load
It means currently applied load.
The range of the value is 0~2047, and the unit is about 0.1%.
If the value is 0~1023, it means the load works to the CCW direction.
If the value is 1024~2047, it means the load works to the CW direction.
That is, the 10th bit becomes the direction bit to control the direction, and 1024 is equal to 0.
For example, the value is 512, it means the load is detected in the direction of CCW about 50% of the maximum torque.

BIT	15~11	10	9	8	7	6	5	4	3	2	1	0
Value	0	Load Direction	Data (Load Ratio)									

Load Direction = 0 : CCW Load, Load Direction = 1: CW Load

Notes: Current load is inferred from the internal torque value, not from Torque sensor etc.
For that reason, it cannot be used to measure weight or torque; however, it must be used only to detect which direction the force works.

Present Voltage
It is the size of the current voltage supplied.
This value is 10 times larger than the actual voltage. For example, when 10V is supplied, the data value is 100 (0x64)

Present Temperature
It is the internal temperature of Dynamixel in Celsius.
Data value is identical to the actual temperature in Celsius. For example, if the data value is 85 (0x55), the current internal temperature is 85° .

Registered Instruction

Value	Meaning
0	There are no commands transmitted by REG_WRITE
1	There are commands transmitted by REG_WRITE.

Notes: If ACTION command is executed, the value is changed into 0.

Moving

Value	Meaning
0	Goal position command execution is completed.
1	Goal position command execution is in progress.

Lock

Value	Meaning
0	EEPROM area can be modified.
1	EEPROM area cannot be modified.

Caution: If Lock is set to 1, the power must be turned off and then turned on again to

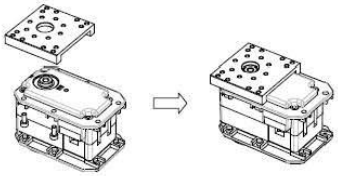
change into 0.

Punch
Current to drive motor is at minimum.
Can choose vales from 0x20 to 0x3FF.

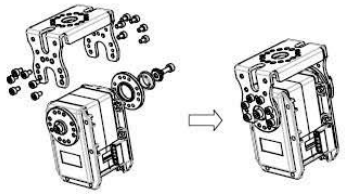
Option Frame(Old Model)

The types of RX-28 option frames are as follows.

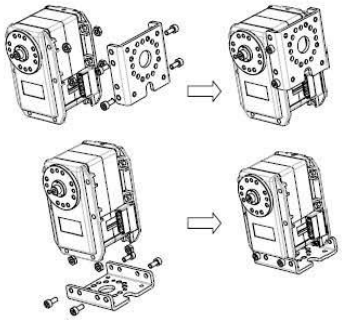
FR07-B1(OF-RX28B)



FR07-H1(OF-RX28H)



FR07-S1(OF-RX28S)

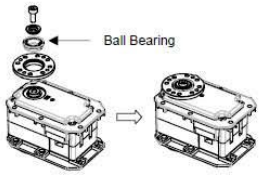


Horn (Old Model)

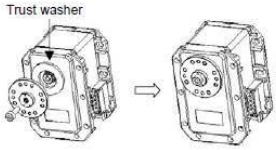
The types of RX-28 Horns are as follows.

HN07-N1(Horn RX-28)

HN07-I1(Bearing Set RX-28H)

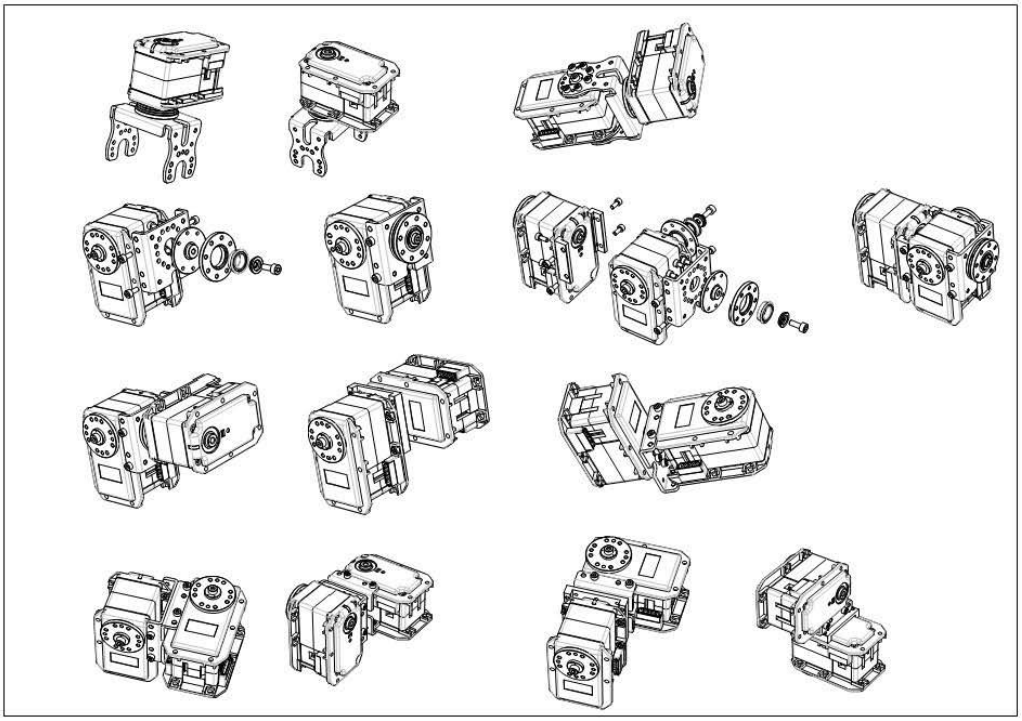


HN07-T1(Horn 28T Set)



Combination

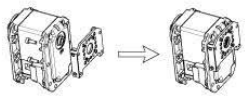
The following example shows the combination structure of option frames and horns.



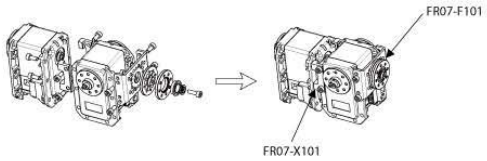
Option Frame (New Model)

The types of RX-28 option frame are as follows.

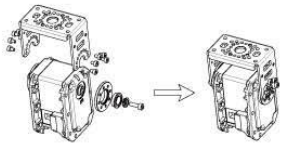
FR07-B101



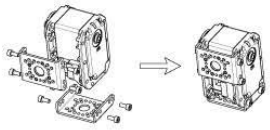
FR07-F101_FR07-X101



FR07-H101



FR07-S101



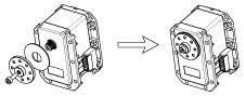
Horn (New Model)

The types of RX-28 Horns are as follows.

HN07-I101

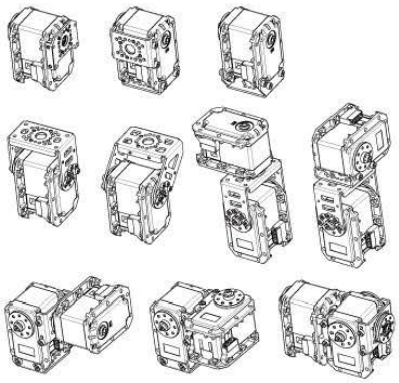
HN07-N101

HN07-T101



Combination (New Model)

The following example shows the combination structure of option frames and horns.



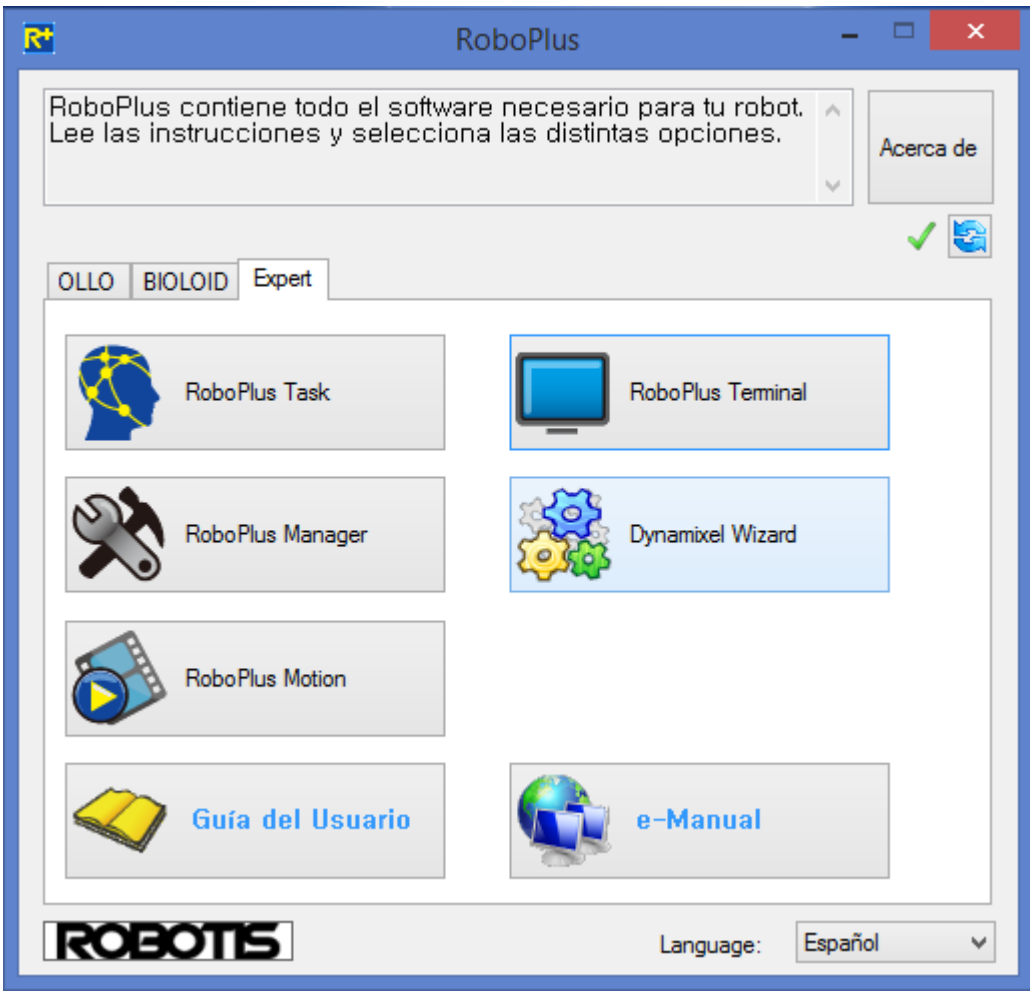
Dimension

Drawing Information : DOWNLOAD RX28Dimension.pdf

APÉNDICE 7

Configuración previa de los servomotores Dynamixel

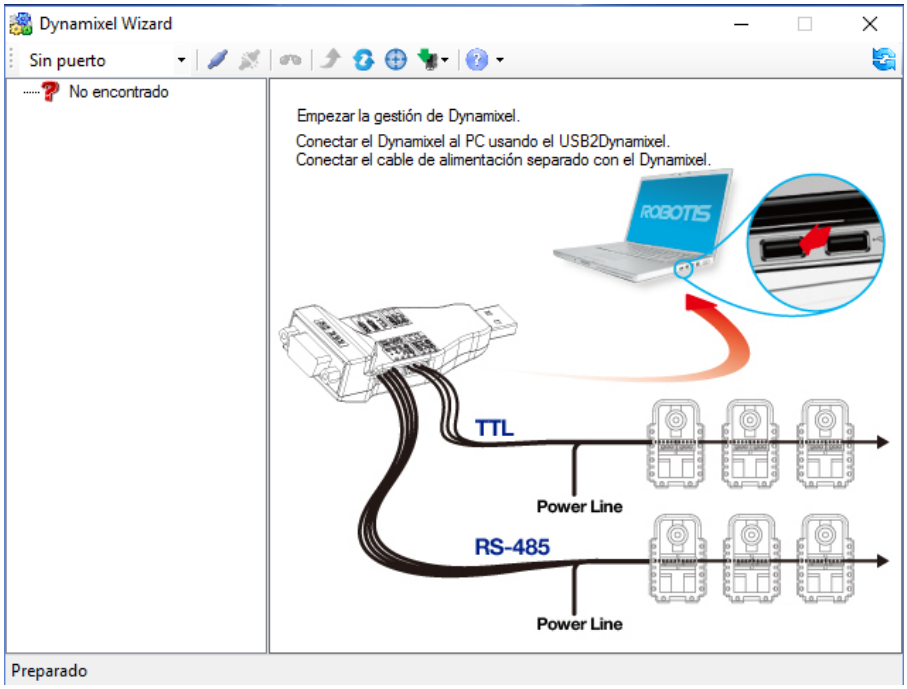
Para configurar el Baudrate delos Dynamixel y el ID que le queremos asignra, haremos uso del un software que proporciona ROBOTIS, llamado “RoboPlus”



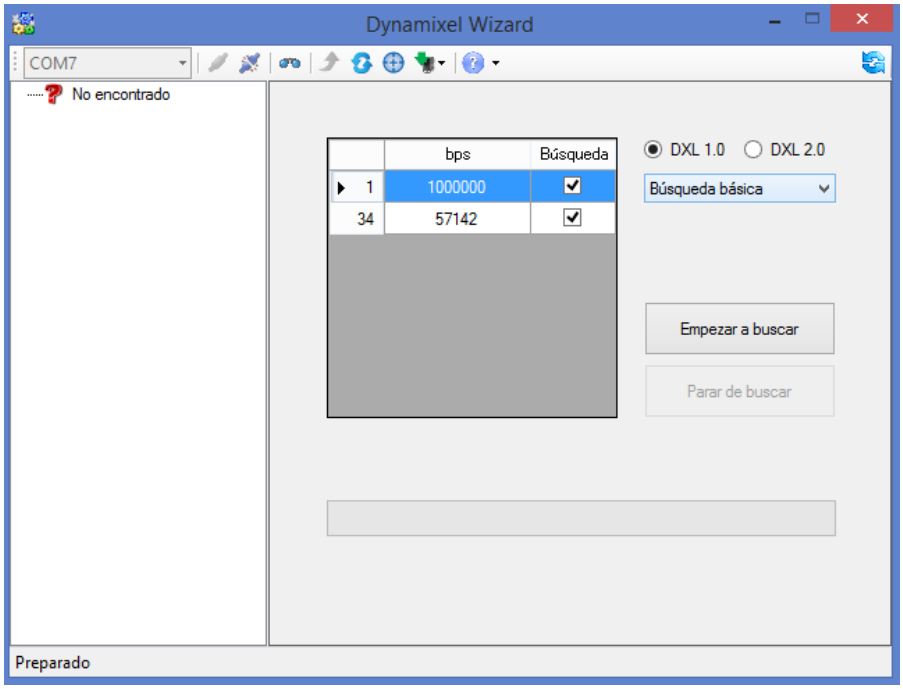
Nos vamos a la pestaña Expert y daremos clic sobre al botón Dynamixel Wizard. Posterormente aparecerá una ventana en donde nos dice que necesitaremos conectar el o los Dynamixel al USB2Dynamixel.

El USB2Dynamixel está estructurado de la siguiente manera (ROBOTIS, 2010b):

Dependiendo la serie de Dynamixel con la que se cuente, se deberán conectar en el 4P Connector (Comunicación RS-485) o en el 3P Connector (Comunicación TTL) como se aprecia en la imagen de ayuda que aparece en la ventana Dynamixel Wizard

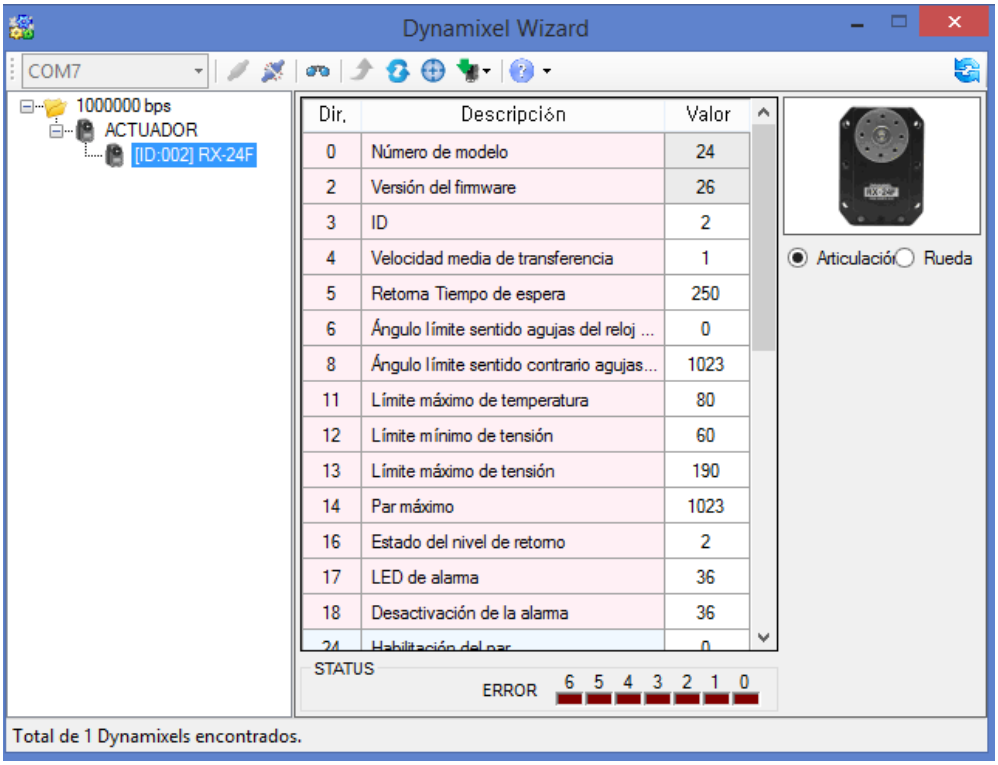


Ya que se tiene los Dynamixel conectados, se selecciona el puerto que la computadora le asigno al USB2Dynamixel y se le pone en Abrir puerto y aparecerá la siguiente ventana:



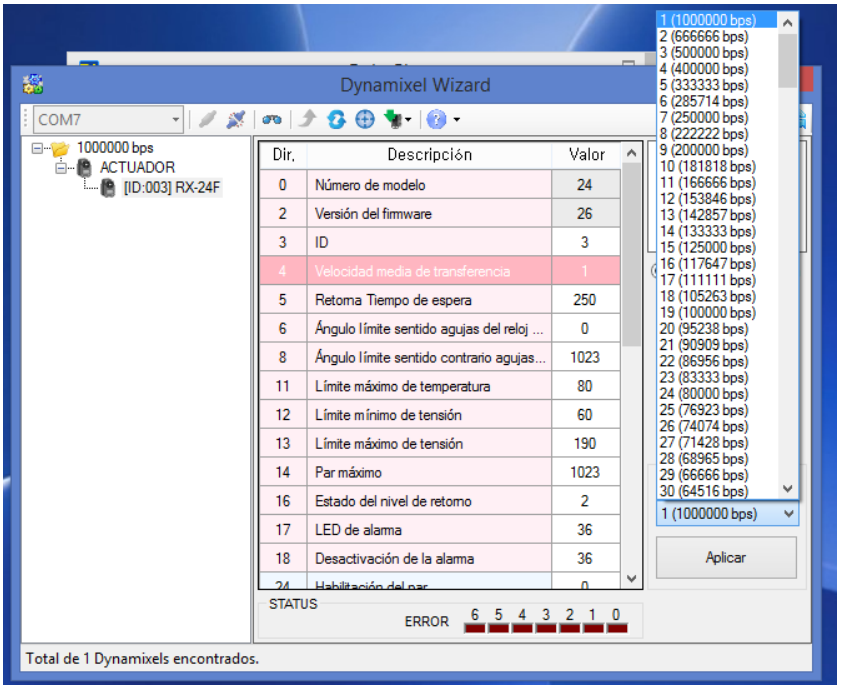
Se dejan los parámetros como aparecen y se le da clic en Empezar a buscar. El software detectará cuantos Dynamixel ha encontrado y aparecerá una lista del lado izquierdo de la ventana con la velocidad de comunicación en la que se encuentra, el ID que tiene configurado y la serie y modelo del Dynamixel.

Ya que termino la búsqueda, para poder cambiar los parámetros del servomotor se le da doble clic sobre uno de los Dynamixel encontrados en donde aparece el ID y la serie y modelo del motor y aparecerá una nueva ventana con una lista de los parámetros que se pueden configurar

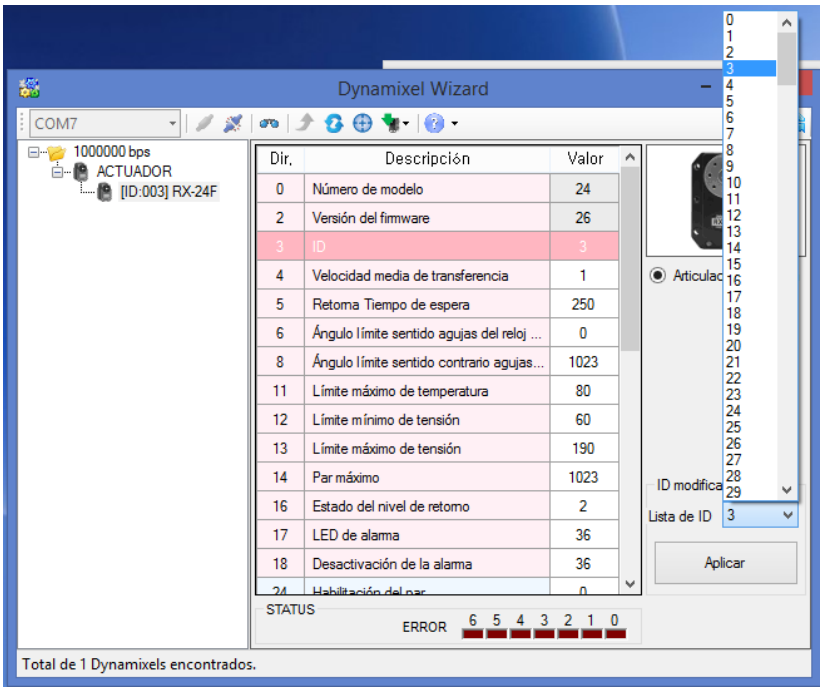


Los datos en rojo, son los parámetros de la Tabla de control EEPROM, que nos sirven de configuración y una vez guardados permanecerán esos datos almacenados, aunque se dejen de alimentar los Dynamixel. De esos parámetros los que nosotros utilizaremos son los de la Velocidad media de transferencia y el ID.

Para lograr la configuración entre Arduino y la librería Dynamixel, es necesario cambiar la Velocidad media de transmisión a 1000000. Para lograrlo, se necesita seleccionar en donde dice valor y velocidad media de transmisión y aparecerá una lista con los posibles valores, seleccionamos el de 1000000 bps y seleccionamos Aplicar.



Para el cambiar el ID, el procedimiento es idéntico al de la velocidad media de transmisión. Aquí podemos seleccionar el ID que nosotros queramos asignarles a los servos conforme más nos convenga.



APÉNDICE 8

Programas para modificar la dirección de la tarjeta MD25

Archivo Editar Sketch Herramientas Ayuda

cambiar_direccion_md25_1 §

```
#include <Wire.h>

#define SOFTWAREREG         0x0D                        // Byte to read the software version, Values of 0 eing sent...
                                                        // ...using write have to be cast as a byte to stop them being...
                                                        // ...misinterperted as NULL
#define SPEED1              0x00                        // Byte to send speed to first motor
#define SPEED2              0x01                        // Byte to send speed to second motor
#define ENCODERONE          0x02                        // Byte to read motor encoder 1
#define ENCODERTWO          0x06                        // Byte to read motor encoder 2
#define VOLTREAD            0x0A                        // Byte to read battery volts
#define RESETENCODERS       0x20
#define CMD                 0x10

byte error, address, nDevices;
byte bandera = 0;

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

void loop()
{
  if (bandera == 0)
    Scan();


 if (bandera == 1)
 {
   //CAMBIO DE DIRECCIÓN I2C
   Wire.beginTransmission(address);
   Wire.write(CMD);
   Wire.write(0xA0);
   Wire.endTransmission();
   Wire.beginTransmission(address);
   Wire.write(CMD);
   Wire.write(0xAA);
   Wire.endTransmission();
   Wire.beginTransmission(address);
   Wire.write(CMD);
   Wire.write(0xA5);
   Wire.endTransmission();
   Wire.beginTransmission(address);
   Wire.write(CMD);
   Wire.write(0xB0);
   Wire.endTransmission();

   Serial.println("\nCambio de direccion a 0x58 (DEFAULT)");
   bandera = 2;
 }
}
```

```
//METODOS
void Scan()
{
  Serial.println("\nI2C Scanner\n");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");

      if (address<16)
        Serial.print("0");

      Serial.print(address,HEX);
      nDevices++;
    }

    else if (error==4)
    {
      Serial.print("Unknow error at address 0x");


      if (address<16)
        Serial.print("0");

      Serial.println(address,HEX);
    }
  }

  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    bandera = 1;
}



      if (address<16)
        Serial.print("0");

      Serial.println(address,HEX);
    }
  }

  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
```

Dato	Address I2C	
	HEX	DEC
B0	0x58	88
B2	0x59	89
B4	0x5A	90
B6	0x5B	91
B8	0x5C	92
BA	0x5D	93
BC	0x5E	94
BE	0x5F	95

Archivo Editar Sketch Herramientas Ayuda

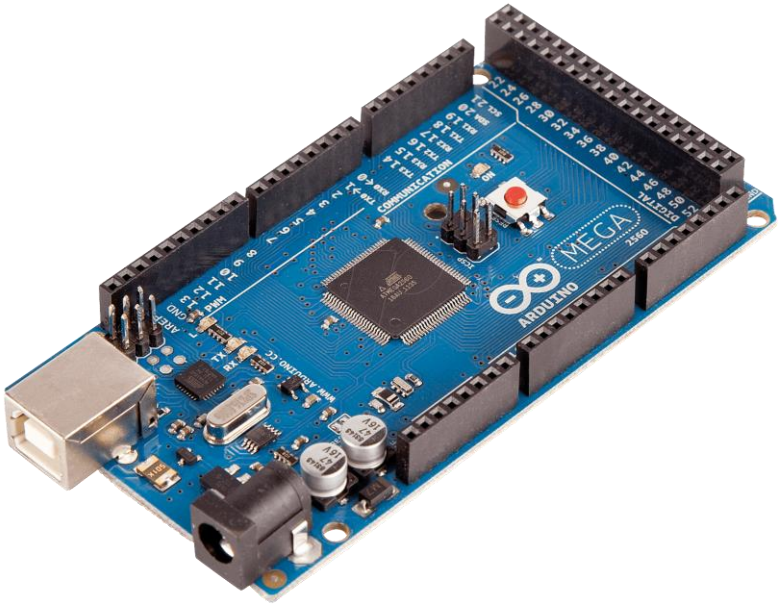
cambiar_direccion_md25_2 §

```
#include <Wire.h>

#define MD25ADDRESS         0x58                        // Address of the MD25

#define SOFTWAREREG         0x0D                        // Byte to read the software version,...
                                                        // Values of 0 eing sent using write have to be...
                                                        // cast as a byte to stop them being misinterperted as NULL
#define SPEED1              0x00                        // Byte to send speed to first motor
#define SPEED2              0x01                        // Byte to send speed to second motor
#define ENCODERONE          0x02                        // Byte to read motor encoder 1
#define ENCODERTWO          0x06                        // Byte to read motor encoder 2
#define VOLTREAD            0x0A                        // Byte to read battery volts
#define RESETENCODERS       0x20
#define CMD                 0x10

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

void loop()
{
  //CAMBIO DE DIRECCIÓN I2C
  Wire.beginTransmission(MD25ADDRESS);
  Wire.write(CMD);
  Wire.write(0xA0);
```

```
  Wire.endTransmission();
  Wire.beginTransmission(MD25ADDRESS);
  Wire.write(CMD);
  Wire.write(0xAA);
  Wire.endTransmission();
  Wire.beginTransmission(MD25ADDRESS);
  Wire.write(CMD);
  Wire.write(0xA5);
  Wire.endTransmission();
  Wire.beginTransmission(MD25ADDRESS);
  Wire.write(CMD);
  Wire.write(0xB2);                  //Se debe poner B0, B2, B4, B6, B8, BA, BC o BE
  Wire.endTransmission();

  Serial.println("Cambio de direccion: OK");
}
```

APÉNDICE 9

Programas de Arduino



CONTROLADOR SIMULINK (DYNAMIXEL)

Archivo Editar Sketch Herramientas Ayuda

Controlador_Dynamixel

```
//TESIS//
#include <DynamixelSeriall.h>

int   posd, posd_val, vel;
byte  id, posd_h, posd_l, vel_h, vel_l;
int   posa, posa_val, Speed, Load, Voltage, Temperature;
byte  posa_h, posa_l, Speed_h, Speed_l, Load_h, Load_l;

byte bandera;

void setup ()
{
  Serial.begin(9600);
  Dynamixel.begin(1000000,2);     // Inicialize the servo at 1Mbps and Pin Control 2
  bandera = 0;
}

void loop ()
{

  if(bandera == 0)
  {
    if(Serial.available()>0)
    {
      id = Serial.read();
      posd_h = Serial.read();
      posd_l = Serial.read();
      vel h = Serial.read();

     vel_l = Serial.read();

     posd = posd_h * 256 + posd_l;
     vel = vel_h * 256 + vel_l;

     bandera=1;
    }
  }

 else if (bandera == 1)
 {

   posd_val = map(posd,0,300,0,1024);
   Dynamixel.moveSpeed (id,posd_val,vel);
```

```
    Speed = Dynamixel.readSpeed(id);
    posa_val = Dynamixel.readPosition(id);
    Load = Dynamixel.readLoad(id);
    Voltage = Dynamixel.readVoltage(id);
    Temperature = Dynamixel.readTemperature(id);

    posa = map(posa_val,0,1024,0,300);
    posa_h = posa>>8;
    posa_l = posa;
    Speed_h = Speed>>8;
    Speed_l = Speed;
    Load_h = Load>>8;

     Load_l = Load;

     bandera = 2;
   }

   else if (bandera == 2)
   {
     Serial.write(id);
     Serial.write(posa_h);
     Serial.write(posa_l);
     Serial.write(Speed_h);
     Serial.write(Speed_l);
     Serial.write(Load_h);
     Serial.write(Load_l);
     Serial.write(Voltage);
     Serial.write(Temperature);

     bandera = 0;
   }

   else
   {
     bandera = 0;
   }
}
```

CONTROLADOR SIMULINK (MD25)

Archivo Editar Sketch Herramientas Ayuda

Controlador_MD25 §

```
//TESIS//
#include <Wire.h>

//#define MD25ADDRESS         0x58    // Address of the MD25

#define SPEED1              0x00    // Register to read or write speed to first motor
#define SPEED2              0x01    // Register to read or write speed to second motor
#define ENCODERONE          0x02    // Register to read motor encoder 1
#define ENCODERTWO          0x06    // Register to read motor encoder 2
#define VOLTREAD            0x0A    // Register to read battery volts
#define M1CURRENT           0x0B    // Register to read current through motor1
#define M2CURRENT           0x0C    // Register to read current through motor2
#define SOFTWAREREV         0x0D    // Register to read the software version, Values of 0 eing sent using ...
                                    // ...write have to be cast as a byte to stop them being misinterperted as NULL
#define ACCELERATION        0x0E    // Register to read or write acceleration
#define MODE                0x0F    // Register to read or write mode of operation speed motors
#define CMD                 0x10    // Command Register (write only)
#define RESETENCODERS       0x20    // Register to reset encoders
#define SPEEDREGULATION     0x31    // Register to enable automatic speed regulation

byte Address, SpeedA, SpeedB;
byte enc1[4], enc2[4], battery, current1, current2;
long pos1, pos2;

int bandera;

void setup()
{

  Wire.begin();
  Serial.begin(9600);
  encodersReset();
  bandera=0;
  delay(100);
}

void loop()
{
  if(bandera == 0)
  {
    if(Serial.available()>0)
    {
      Address = Serial.read();
      SpeedA = Serial.read();
      SpeedB = Serial.read();

      bandera=1;
    }
  }

  else if (bandera == 1)
  {
    Move(Address,SpeedA,SpeedB);
```

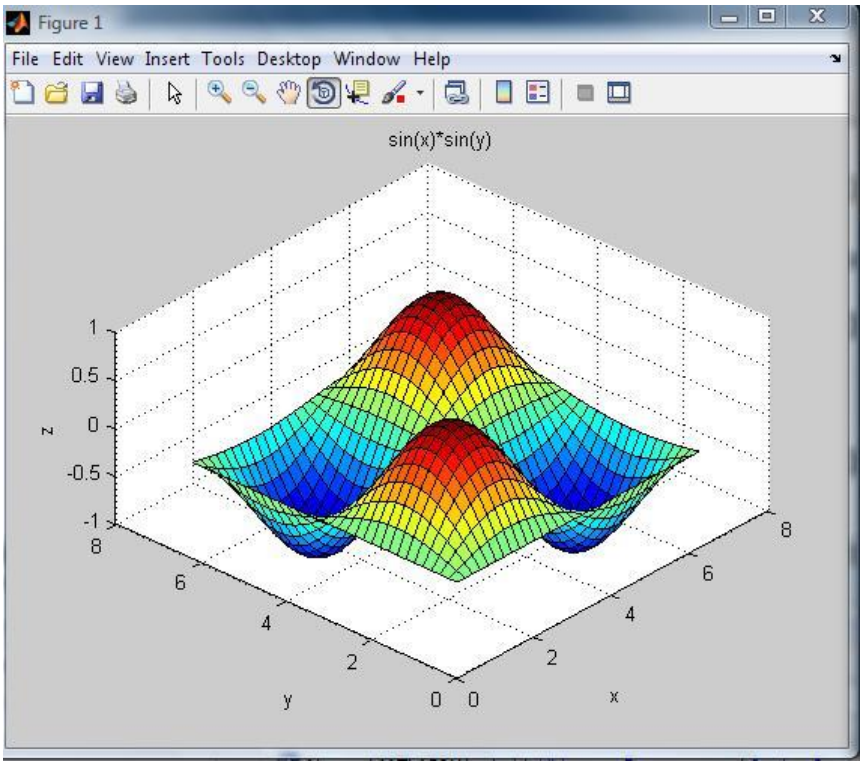
```
    enc_1();
    enc_2();
    volts();
    m1current();
    m2current();

    bandera = 2;
  }

  else if (bandera == 2)
  {
    Serial.write(Address);
    Serial.write(enc1[0]);
    Serial.write(enc1[1]);
    Serial.write(enc1[2]);
    Serial.write(enc1[3]);
    Serial.write(enc2[0]);
    Serial.write(enc2[1]);
    Serial.write(enc2[2]);
    Serial.write(enc2[3]);
    Serial.write(battery);
    Serial.write(current1);
    Serial.write(current2);

    bandera = 0;
  }

  else
  {
    bandera = 0;
  }
}

//METODOS//

void Move(byte Address, byte SpeedA, byte SpeedB)
{
  Wire.beginTransmission(byte(Address));                    // Drive motor 2 at speed value stored in x
  Wire.write(SPEED1);
  Wire.write(SpeedA);
  Wire.endTransmission();

  Wire.beginTransmission(byte(Address));                    // Drive motor 1 at speed value stored in x
  Wire.write(SPEED2);
  Wire.write(SpeedB);
  Wire.endTransmission();
}

void encodersReset()
{
  Wire.beginTransmission(byte(Address));
  Wire.write(CMD);
  Wire.write(RESETENCODERS);
```

```
  Wire.endTransmission();
}

void enc_1()
{
  Wire.beginTransmission(byte(Address));
  Wire.write(ENCODERONE);
  Wire.endTransmission();

  Wire.requestFrom(byte(Address), 4);
  while(Wire.available() < 4);

  possl = Wire.read();
  possl <<= 8;
  possl += Wire.read();
  possl <<= 8;
  possl += Wire.read();
  possl <<= 8;
  possl += Wire.read();

  encl[0] = possl;
  encl[1] = possl>>8;
  encl[2] = possl>>16;
  encl[3] = possl>>24;
}
void enc_2()
{
  Wire.beginTransmission(byte(Address));
  Wire.write(ENCODERTWO);
  Wire.endTransmission();

  Wire.requestFrom(byte(Address), 4);
  while(Wire.available() < 4);

  poss2 <<= 8;
  poss2 += Wire.read();
  poss2 <<= 8;
  poss2 += Wire.read();
  poss2 <<= 8;
  poss2 += Wire.read();

  enc2[0] = poss2;
  enc2[1] = poss2>>8;
  enc2[2] = poss2>>16;
  enc2[3] = poss2>>24;
}

void volts()
{
  Wire.beginTransmission(byte(Address));
  Wire.write(VOLTREAD);
  Wire.endTransmission();


  Wire.requestFrom(byte(Address), 1);
  while(Wire.available() < 1);

  battery = Wire.read();
}

void mlcurrent()
{
  Wire.beginTransmission(byte(Address));
  Wire.write(M1CURRENT);
  Wire.endTransmission();
```

```
  Wire.requestFrom(byte(Address), 1);
  while(Wire.available() < 1);

  current1 = Wire.read();
}

void m2current()
{
  Wire.beginTransmission(byte(Address));
  Wire.write(M2CURRENT);
  Wire.endTransmission();

  Wire.requestFrom(byte(Address), 1);
  while(Wire.available() < 1);

  current2 = Wire.read();
}
```

APÉNDICE 10

Script para generar gráficas



Graficas_Dynamixel.m

```
nombre =  'Prueba1.xlsx';

%Dynamixel ID1
%Posicion actual vs Posicion leida
figure(2)

datos = xlsread(nombre,3);
l1=length(datos);
posE = datos(1:l1,1);
t1=(l1-1)/10;
t1=0:0.1:t1;

datos = xlsread(nombre,7);
l2=length(datos);
posS = datos(1:l2,1);
t2=(l2-1)/10;
t2=0:0.1:t2;

plot(t1,posE, '-b*')
hold on
plot(t2,posS, '-r*')

title('Dynamixel ID 1')
xlabel('Tiempo [t]')
ylabel('Posición [°]')
grid on

%Velocidad leida
figure(3)

datos = xlsread(nombre,7);
l=length(datos);
vel = datos(1:l,2);
t=(l-1)/10;
t=0:0.1:t;

plot(t,vel, '-b*')

title('Dynamixel ID 1')
xlabel('Tiempo [t]')
ylabel('Velocidad [RPM]')
grid on

%Porcentaje de carga
figure(4)

datos = xlsread(nombre,7);
l=length(datos);
carga = datos(1:l,3);
t=(l-1)/10;
t=0:0.1:t;

plot(t,carga, '-b*')

title('Dynamixel ID 1')
xlabel('Tiempo [t]')
ylabel('Carga [%]')
grid on

%Sentido
figure(5)

datos = xlsread(nombre,7);
l=length(datos);
sentido = datos(1:l,4);
t=(l-1)/10;
t=0:0.1:t;

plot(t,sentido, '-b*')

title('Dynamixel ID 1')
xlabel('Tiempo [t]')
ylabel('Sentido de giro [1-CCW 0-CW]')
grid on

%Voltaje
figure(6)

datos = xlsread(nombre,7);
l=length(datos);
volt = datos(1:l,5);
t=(l-1)/10;
t=0:0.1:t;

plot(t,volt, '-b*')

title('Dynamixel ID 1')
xlabel('Tiempo [t]')
ylabel('Voltaje [V]')
grid on

%Temperatura
figure(7)

datos = xlsread(nombre,7);
l=length(datos);
temp = datos(1:l,6);
t=(l-1)/10;
```

```
t=0:0.1:t;

plot(t,temp, '-b*')

title('Dynamixel ID 1')
xlabel('Tiempo [t]')
ylabel('Temperatura [°C]')
grid on

%Dynamixel ID2
%Posicion actual vs Posicion leida
figure(8)

datos = xlsread(nombre,4);
l1=length(datos);
posE = datos(1:l1,1);
t1=(l1-1)/10;
t1=0:0.1:t1;

datos = xlsread(nombre,8);
l2=length(datos);
posS = datos(1:l2,1);
t2=(l2-1)/10;
t2=0:0.1:t2;

plot(t1,posE, '-b*')
hold on
plot(t2,posS, '-r*')

title('Dynamixel ID 2')
xlabel('Tiempo [t]')
ylabel('Posición [°]')
grid on

%Velocidad leida
figure(9)

datos = xlsread(nombre,8);
l=length(datos);
vel = datos(1:l,2);
t=(l-1)/10;
t=0:0.1:t;

plot(t,vel, '-b*')

title('Dynamixel ID 2')
xlabel('Tiempo [t]')
ylabel('Velocidad [RPM]')
grid on

%Porcentaje de carga
figure(10)

datos = xlsread(nombre,8);
l=length(datos);
carga = datos(1:l,3);
t=(l-1)/10;
t=0:0.1:t;

plot(t,carga, '-b*')

title('Dynamixel ID 2')
xlabel('Tiempo [t]')
ylabel('Carga [%]')
grid on

%Sentido
figure(11)

datos = xlsread(nombre,8);
l=length(datos);
sentido = datos(1:l,4);
t=(l-1)/10;
t=0:0.1:t;

plot(t,sentido, '-b*')

title('Dynamixel ID 2')
xlabel('Tiempo [t]')
ylabel('Sentido de giro [1-CCW 0-CW]')
grid on

%Voltaje
figure(12)

datos = xlsread(nombre,8);
l=length(datos);
volt = datos(1:l,5);
t=(l-1)/10;
t=0:0.1:t;

plot(t,volt, '-b*')

title('Dynamixel ID 2')
xlabel('Tiempo [t]')
ylabel('Voltaje [V]')
grid on

%Temperatura
figure(13)

datos = xlsread(nombre,8);
```

```
l=length(datos);
temp = datos(1:l,6);
t=(l-1)/10;
t=0:0.1:t;

plot(t,temp, '-b*')

title('Dynamixel ID 2')
xlabel('Tiempo [t]')
ylabel('Temperatura [°C]')
grid on

%Dynamixel ID3
%Posicion actual vs Posicion leida
figure(14)

datos = xlsread(nombre,5);
l1=length(datos);
posE = datos(1:l1,1);
t1=(l1-1)/10;
t1=0:0.1:t1;

datos = xlsread(nombre,9);
l2=length(datos);
posS = datos(1:l2,1);
t2=(l2-1)/10;
t2=0:0.1:t2;

plot(t1,posE, '-b*')
hold on
plot(t2,posS, '-r*')

title('Dynamixel ID 3')
xlabel('Tiempo [t]')
ylabel('Posición [°]')
grid on

%Velocidad leida
figure(15)

datos = xlsread(nombre,9);
l=length(datos);
vel = datos(1:l,2);
t=(l-1)/10;
t=0:0.1:t;

plot(t,vel, '-b*')

title('Dynamixel ID 3')
xlabel('Tiempo [t]')
ylabel('Velocidad [RPM]')
grid on

%Porcentaje de carga
figure(16)

datos = xlsread(nombre,9);
l=length(datos);
carga = datos(1:l,3);
t=(l-1)/10;
t=0:0.1:t;

plot(t,carga, '-b*')

title('Dynamixel ID 3')
xlabel('Tiempo [t]')
ylabel('Carga [%]')
grid on

%Sentido
figure(17)

datos = xlsread(nombre,9);
l=length(datos);
sentido = datos(1:l,4);
t=(l-1)/10;
t=0:0.1:t;

plot(t,sentido, '-b*')

title('Dynamixel ID 3')
xlabel('Tiempo [t]')
ylabel('Sentido de giro [1-CCW 0-CW]')
grid on

%Voltaje
figure(18)

datos = xlsread(nombre,9);
l=length(datos);
volt = datos(1:l,5);
t=(l-1)/10;
t=0:0.1:t;

plot(t,volt, '-b*')

title('Dynamixel ID 3')
xlabel('Tiempo [t]')
ylabel('Voltaje [V]')
grid on

%Temperatura
```

```
figure(19)

datos = xlsread(nombre,9);
l=length(datos);
temp = datos(1:l,6);
t=(l-1)/10;
t=0:0.1:t;

plot(t,temp, '-b*')

title('Dynamixel ID 3')
xlabel('Tiempo [t]')
ylabel('Temperatura [°C]')
grid on
```

Graficas_MD25

```
nombre =  'PruebaMD25.xlsx';

%Valores enviados a la MD25
figure(1)

datos = xlsread(nombre,2);
l=length(datos);
motA = datos(1:l,1);

datos = xlsread(nombre,2);
l=length(datos);
motB = datos(1:l,2);

t=(l-1);
t=0:1:t;

plot(t,motA, '-b*')
hold on
plot(t,motB, '-r*')

title('Velocidad motores A y B')
xlabel('Lecturas')
ylabel('Valores')
grid on

%Lectura de Encoder Motor A
figure(2)

datos = xlsread(nombre,2);
l=length(datos);
encA = datos(1:l,4);

plot(t,encA, '-b*')

title('Encoder Motor A')
xlabel('Lecturas')
ylabel('Valores encoder')
grid on

%Lectura de Encoder Motor B
figure(3)

datos = xlsread(nombre,2);
l=length(datos);
encB = datos(1:l,5);

plot(t,encB, '-b*')

title('Encoder Motor B')
xlabel('Lecturas')
ylabel('Valores encoder')
grid on

%Lectura de Voltaje
figure(4)

datos = xlsread(nombre,2);
l=length(datos);
v = datos(1:l,7);

plot(t,v, '-b*')

title('Voltaje MD25')
xlabel('Lecturas')
ylabel('Voltaje [V]')
grid on

%Lectura de Corriente Motor A
figure(5)

datos = xlsread(nombre,2);
l=length(datos);
iA = datos(1:l,9);

plot(t,iA, '-b*')

title('Corriente Motor A')
xlabel('Lecturas')
ylabel('Corriente [A]')
grid on

%Lectura de Corriente Motor B
figure(6)

datos = xlsread(nombre,2);
l=length(datos);
iB = datos(1:l,11);

plot(t,iB, '-b*')

title('Corriente Motor B')
xlabel('Lecturas')
ylabel('Corriente [A]')
grid on
```