


UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA

INGENIERÍA EN AUTOMATIZACIÓN

Desarrollo de un controlador de movimiento para un robot humanoide

TESIS

Que como parte de los requisitos para obtener el título de Ingeniero en Automatización con Línea Terminal en Mecatrónica.

Presenta: **Arturo Iram de la Fuente Sánchez**

Director: **Dr. Gerardo Israel Pérez Soto**

Co-Directora: **Dra. Karla Anhel Camarillo Gómez**

Presidente:
Dr. Gerardo Israel Pérez Soto _____ **Firma**

Secretario:
Dra. Karla Anhel Camarillo Gómez _____ **Firma**

Vocal:
Dr. Jesús Rooney Rivera Guillén _____ **Firma**

Sinodal:
Dr. Carlos Gustavo Manríquez Padilla _____ **Firma**

Sinodal:
Dr. Juvenal Rodríguez Reséndiz _____ **Firma**

Querétaro, Qro.

Mayo 2023

Índice

Índice de figuras

Índice de tablas

1. Introducción

1.1. Antecedentes

A lo largo de la historia, desde el origen de la humanidad, el hombre se encuentra continuamente en constante evolución y cambio, cada actividad que realiza sufre algún tipo de modificación, ya sea en su forma de desempeñar cierta actividad o en los materiales que se utilizan para la misma. Esto sugiere que conforme el hombre descubre, experimenta y aprende cosas nuevas, la necesidad de evolucionar y mejorar, siempre se encuentra presente. Es esta curiosidad y la constante evolución lo que dio origen a lo que hoy en día conocemos como tecnología, con el desarrollo de la primera herramienta desarrollada por el hombre primitivo.

Conforme la tecnología evoluciona, el hombre encuentra fascinación por los mecanismos, máquinas y dispositivos complejos, los cuales, dependiendo de su construcción, podían tener objetivos y funcionalidades diferentes, por ejemplo, el mecanismo de anticitera, un dispositivo considerado una "computadora astronómica" diseñado para calcular la posición de los astros, construido en el periodo helenístico, aproximadamente en el 70 a.C. [4].

Uno de los motivos principales en el desarrollo de dispositivos cada vez más complejos, es facilitar las tareas del hombre, comenzando con aquellas que representen trabajos peligrosos y/o incesantes. Dichos dispositivos complejos dieron inicio a lo que, con el paso del tiempo se denominó "robot". La palabra robot se utiliza por primera vez en 1921, en la obra de teatro Rossum's Universal Robots [5], escrita por el checo Karel Čapek (1890-1938) , sin embargo, la palabra robot, así como el término "robótica" fueron impulsados gracias al escritor Isaac Asimov (1920-1992) debido a su obra Runaround publicada en 1942. Ambos términos se encuentran en constante cambio debido al dinamismo que tienen. Hace algunos años, artefactos que solo existían en la ciencia ficción actualmente se están volviendo una realidad tecnológica dentro de lo que ahora se conoce como el área de la robótica.

Debido al dinamismo mencionado anteriormente, definir de manera precisa el término "robot" se vuelve una tarea compleja gracias al basto campo de dispositivos que entran dentro de esta categoría, algunas de las definiciones más comunes serían:

- *Máquina o ingenio electrónico programable que es capaz de manipular objetos y realizar diversas operaciones* [6].
- *Cualquier máquina operada automáticamente que sustituye el esfuerzo humano, no tiene por qué tener apariencia humana o desarrollar funciones de la manera en como lo realizan los humanos* [7].

Sin embargo, estas definiciones aún resultan ser insuficientes para abarcar de manera completa los dispositivos que son considerados "robots". Algunos de estos dispositivos son los robots caminantes, robots aéreos, robots manipuladores, robots de entretenimiento, robots humanoides, entre otros.

Un robot humanoide es un dispositivo diseñado con el objetivo de simular la forma y movimientos de un ser humano, generalmente dotados de un torso, dos extremidades superiores y dos extremidades inferiores [8]. Actualmente, la popularidad de los robots provoca que el ingreso de estos a la vida cotidiana de los humanos, sea cada vez más frecuente. Esto lleva a preguntas de investigación desafiantes como la aceptación de un robot por parte del usuario humano, la seguridad de los humanos que interactúan con el robot, el sistema cognitivo y la inteligencia artificial requeridas del sistema robótico [9]. Claramente para los robots humanoides esto no es excepción, las investigaciones sobre robots humanoides colaborativos son una realidad, sin embargo, es más común observar robots humanoides pequeños diseñados para fines competitivos o para fines educacionales. Gracias a la popularidad de competencias de robots humanoides, existen actualmente un gran y creciente número de investigadores que tienen acceso a plataformas físicas de robots humanoides, debido a la facilidad de obtención de dispositivos comerciales distribuidos para este fin [10].

Dentro de las competencias de robots humanoides, una de las más importantes a nivel internacional es la *Federación de Asociaciones Deportivas Internacionales*, FIRA por sus siglas en ingles fundada por el Prof. Jong-Hwan Kim, KAIST, Corea en 1996, es la competencia de fútbol de robots más antigua del mundo y hoy en día, también es una de las competencias más importantes de robótica con el objetivo de utilizar los deportes como problemas de referencia para la investigación de vanguardia en robótica y otras áreas relacionadas [11]. Actualmente cuenta con diferentes categorías y subcompetencias que se llevan a cabo de manera presencial en diferentes sedes alrededor del mundo, sin

embargo, debido a la pandemia del Coronavirus (COVID-19), del año 2020 al 2022 se realiza de manera virtual.

Dentro de la página oficial de FIRA, es posible observar que uno de sus asociados es la empresa ROBOTIS, la cual, es proveedora a nivel global de soluciones robóticas, además de ser uno de los principales fabricantes de hardware robótico utilizado en todos los campos de estudio y en la industria. La empresa cuenta con la marca DYNAMIXEL la cual se encarga de la producción de servomotores inteligentes denominados "todo en uno" [12]. La empresa proporciona una variedad de robots humanoides, cada uno dotado de sus respectivos motores, piezas, accesorios y controladores.

En el presente trabajo, se desarrolla un controlador de movimiento para un robot humanoide BIOLOID Premium tipo A de la marca ROBOTIS que mediante el empleo del microcontrolador Raspberry Pi Pico se abre una forma diferente de programación y control sobre los movimientos del robot.

1.2. Justificación

Un robot humanoide por el hecho de contener un torso, dos extremidades superiores y dos extremidades inferiores (en algunos casos también una cabeza móvil) requiere tener un gran número de grados de libertad con la finalidad de obtener movimiento lo más aproximado al movimiento de un humano. La forma en como se transmite movimiento a cada grado de libertad es por medio de algún dispositivo que proporcione potencia mecánica, lo más común es el empleo de servomotores. Esto conlleva entonces a la implementación de un sistema de control que sea capaz de proporcionar lo necesario para el correcto funcionamiento y movimiento de cada grado de libertad.

Hoy en día, en el mercado existen dispositivos capaces de proporcionar la potencia necesaria para realizar con éxito la tarea mencionada anteriormente. Revisando los controladores de ROBOTIS el principal fabricante y distribuidor de hardware robótico, en la Tabla 1 se presentan algunos de estos controladores, es posible rescatar algunas características similares entre ellos.

Dispositivo	CPU	Módulos de comunicación	Sensores	I/O externo	Costo (USD)
OpenCR	STM32F746ZGT6 32-bit	USB, TTL, RS485, UARTx2, CAN	MPU9250	32 pines - arduino, Módulo sensor 4 pines, conector de extensión de 18 pines	$214.9
CM-700	ATMega 2561	-	Temperatura, voltaje	TTLx4, RS-485x5	$143.2
CM-550	ARM Cortex-M4	Modulo esclavo BLE	Temperatura, voltaje, acelerómetro	Puerto 5 pines x5, Puertos Dynamixel x6	$131.3
CM-530	ARM Cortex STM32F103RE	-	Voltaje	Puerto 5 pines x6, Puertos Dynamixel TTL x5	$119.4

Tabla 1: Tabla comparativa de controladores.

La Tabla 1 muestra diferentes controladores de la misma marca, cada uno con costos y características diferentes. Una de las principales desventajas que se observa en la mayoría de estos controladores, recae en los pines de entrada y salida externos que proporcionan debido a que estos no son realmente útiles para la conexión con dispositivos comúnmente utilizados, un ejemplo, es la posibilidad de conectar dispositivos que requieran puertos digitales. Es posible adentrarse en las características generales de estos controladores debido a que son proporcionadas por el fabricante, por lo cual, se puede observar que la mayoría de los pines externos de entrada y/o salida son utilizados exclusivamente para conexión de dispositivos propios de la marca, limitando casi por completo la conexión con instrumentos de fácil acceso de marcas ajenas al fabricante de los controladores. Otra desventaja son los costos, aunque el precio es un tema relativo, la realidad es que existen dispositivos que, ensamblados de manera correcta, tienen un costo significativamente menor y que son capaces de realizar el trabajo llevado a cabo por los controladores de la Tabla 1. Un ejemplo de esto es el microcontrolador Raspberry Pi Pico, el cual se implementará en la realización de este trabajo de tesis.

En el presente trabajo se realizará el diseño de un controlador de movimientos para un robot humanoide, utilizando un microcontrolador Raspberry Pi Pico, abriendo la posibilidad de añadir dispositivos externos de diversos fabricantes y disminuir a su vez el costo del controlador.

1.3. Descripción del problema

El sistema actual que provee la empresa ROBOTIS para el control y programación de los robots humanoides BIOLOID Premium tipo A incluye un controlador CM-530, en la Tabla 2 se muestran sus especificaciones.

Como anteriormente se menciono en la Sección 1.2, queda prácticamente inhabilitada la escalabilidad en la implementación de dispositivos externos para aumentar la robustez del sistema debido a las limitaciones de hardware y software del controlador proporcionado por el fabricante.

Actualmente, los robots humanoides utilizados para competencias, cuentan principalmente con dos recursos importantes extra, además del controlador, los cuales son, un sistema de visión y una unidad de medición inercial IMU por sus siglas en ingles. En el 2006, Behnke [13] muestra un ejemplo. Volviendo nuevamente a la Tabla 2, el controlador CM-530 es un controlador de arquitectura cerrada ya que el fabricante no proporciona información detallada de la estructura del sistema, por ejemplo el esquemático del mismo. Para un desarrollador es necesario tener más control sobre el comportamiento de un robot humanoide y claramente, el hecho de que el controlador mencionado presente estas características, limita la capacidad de proveer más funcionalidades a un robot humanoide [14].

Característica	Descripción
Peso	54g
CPU	STM32F103RE
Voltaje de alimentación	6v - 15v
GPIO interno	5 botones; micrófono; sensor de voltaje
GPIO externo	6 puertos 5-pin ROBOTIS; 5 conectores DYNAMIXEL series AX/MX

Tabla 2: Especificaciones del controlador CM-530 [2].

En el presente trabajo se pretende realizar el diseño de un controlador de movimientos de arquitectura abierta para un robot humanoide, abriendo la posibilidad de la expansión hacia más funcionalidades.

1.4. Hipótesis

Mediante la combinación de dispositivos comerciales en la región, se puede desarrollar una metodología de control, basada en las características de un sistema de arquitectura abierta, para la interacción de elementos mecánicos, eléctricos y electrónicos que componen a un robot humanoide permitiendo proponer un sistema que ejecute con éxito los respectivos movimientos del robot.

1.5. Objetivos

En esta sección se presenta el objetivo general y los objetivos específicos del presente trabajo de tesis.

1.5.1. Objetivo general

Desarrollar un controlador de movimiento para un robot humanoide empleando un microcontrolador Raspberry Pi Pico.

1.5.2. Objetivos específicos

- Desarrollar tarjeta de potencia para las diferentes alimentaciones del sistema empleando el uso de circuitos reguladores de voltaje y dispositivos de protección.
- Desarrollar la tarjeta de comunicación entre el microcontrolador Raspberry Pi Pico y los servomotores del robot humanoide utilizando dispositivos capaces de realizar conversión de modos de transmisión de datos.
- Implementar un algoritmo de comunicación dentro del microcontrolador para el control de los servomotores utilizando el lenguaje de programación: Micropython.
- Programar los movimientos para el robot humanoide por medio de la nueva interfaz.
- Verificar el funcionamiento del sistema así como de la generación adecuada y eficaz de movimientos poniendo en marcha diferentes rutinas de movimiento.

2. Fundamentos teóricos

En esta sección se describen de forma general los fundamentos teóricos necesarios para la realización del presente trabajo de tesis.

2.1. Robot humanoide BIOLOID Premium tipo A

El robot utilizado en el presente proyecto está basado en el robot humanoide BIOLOID Premium tipo A de la marca ROBOTIS, ver Figura 1.

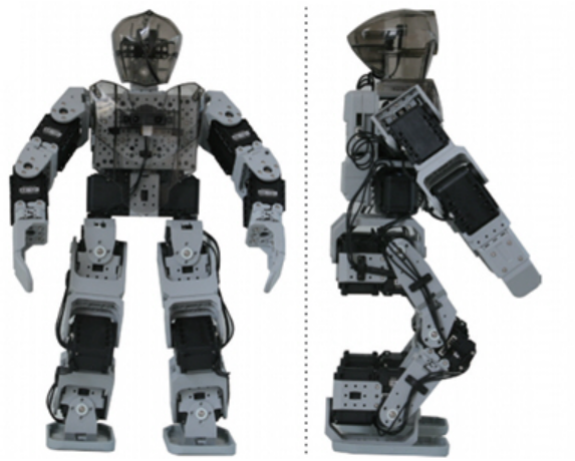


Figura 1: Robot humanoide BIOLOID Premium tipo A [1].

Este robot cuenta con 18 grados de libertad, 12 de ellos en la parte inferior (6 para cada pierna) y 6 en la parte superior (3 para cada brazo), a cada articulación le corresponde un motor Dynamixel. El robot mostrado en la Figura 1 representa el armado que el fabricante aconseja sin embargo, este puede ser modificado a conveniencia del usuario, de igual forma, se encuentra abierta la posibilidad de añadir más grados de libertad conforme se requiera.

2.2. Motores Dynamixel Ax-12a

Los motores para robots de la serie Dynamixel son actuadores modulares que cuentan con una caja reductora de engranajes, un motor DC de precisión y un circuito de control con funcionalidad de red. Tienen la capacidad de detectar y funcionar sobre condiciones

internas como cambios en la temperatura interna o voltaje de suministro. En la Tabla 3 se muestran las características más importantes.

Resolución	**Ángulo de operación**	**Protocolo de comunicación**	**Velocidad de comunicación**
0,29°	300°	Comunicación serial asíncrona Half-duplex.	7843bps - 1Mbps

Tabla 3: Características de motor Dynamixel Ax-12a.

2.2.1. Protocolo de comunicación

El controlador principal se comunica con los motores Dynamixel enviando y recibiendo paquetes de datos. Se tienen dos tipos de paquetes de datos, los paquetes de instrucciones, enviados por el controlador principal hacia los motores y, los paquetes de estado, enviados por los motores hacia el controlador.

En la Figura 2, se muestra el diagrama que ilustra la forma en la que el controlador principal se comunica con múltiples motores a la vez.

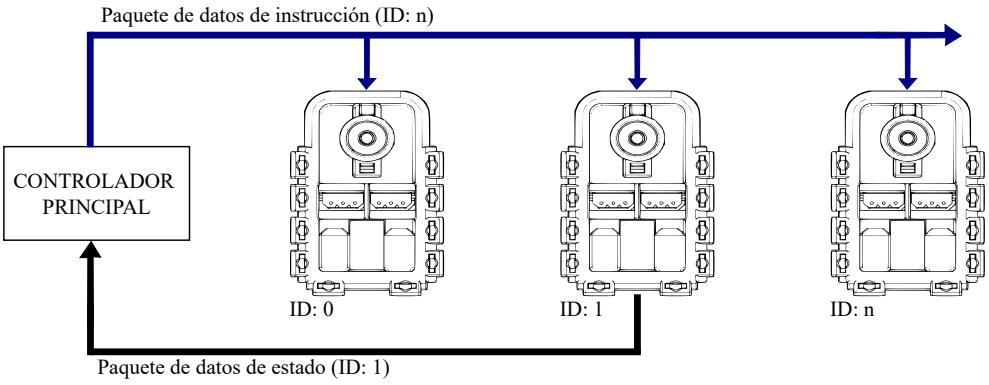


Figura 2: Comunicación entre controlador principal y múltiples motores.

De la Figura 2 se observa que los ID son únicos e irrepetibles en cada motor, por lo que, si existen motores con ID repetido, existirán errores de comunicación.

2.2.2. Paquete de datos de instrucción

Este paquete de datos es enviado por el controlador principal hacia los motores, dicho paquete es obligatorio y necesario para la correcta comunicación entre el controlador

principal y los motores. En la Tabla 4 se muestra la estructura de dicho paquete.

0xFF	0xFF	ID	LONGITUD	INSTRUCCIÓN	PARÁMETRO 1	...	PARÁMETRO N	CHECKSUM

Tabla 4: Paquete de datos de instrucción obligatorios para la comunicación.

A continuación, se describe cada byte que compone el paquete de datos de instrucción:

- **0xFF**: Los dos bytes que contienen el dato 0xFF indica el comienzo del envío de un paquete de datos.
- **ID**: Indica el ID único de cada motor. El rango de ID posible es desde 0 hasta 253.
- **LONGITUD**: Indica el tamaño en byte de la instrucción, parámetros y checksum.
- **INSTRUCCIÓN**: Representa la instrucción a realizar.
- **PARÁMETRO 0..N**: Utilizado si se necesita enviar información adicional además de la instrucción en sí.
- **CHECKSUM**: Es un dato necesario dentro del paquete de datos, el cual, se calcula como: CHECKSUM = !(ID + LONGITUD + INSTRUCCIÓN + PARÁMETRO 1...N).

2.2.3. Paquete de datos de estado

Este paquete es enviado por los motores Dynamixel hacia el controlador después de recibir un paquete de instrucción. La Tabla 5 muestra la estructura del paquete.

0xFF	0xFF	ID	LONGITUD	ERROR	PARÁMETRO 1	...	PARÁMETRO N	CHECKSUM

Tabla 5: Paquete de datos de estado, representan la respuesta de los motores.

Las columnas de la Tabla 5 representan la misma información que la Tabla 4, con excepción de la columna **ERROR**. Este dato representa el byte de error, en caso de existir.

El manual del fabricante [3], permite visualizar a detalle los respectivos comandos de instrucción y de error.

2.3. Microcontrolador Raspberry Pi Pico

Raspberry Pi Pico se conoce como placa de desarrollo de microcontrolador, esto significa que es una placa de circuito impreso que alberga un tipo especial de procesador diseñado para computación física: el microcontrolador. Raspberry Pi Pico está diseñado para proyectos de computación física donde controla cualquier cosa, desde LED y botones hasta sensores, motores e incluso otros microcontroladores [15]. La Figura 3 muestra el microcontrolador en cuestión.

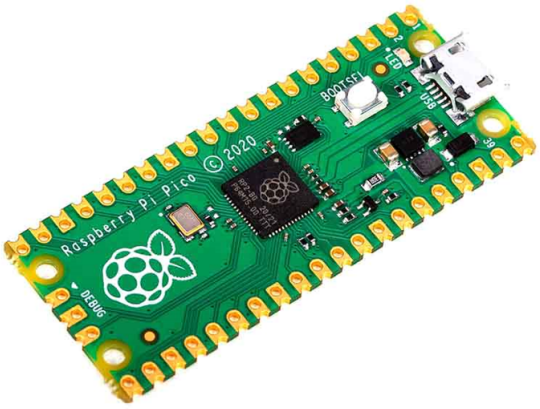


Figura 3: Vista superior de microcontrolador Raspberry Pi Pico.

Las características más importantes de este microcontrolador, se listan a continuación:

- Microcontrolador RP2040 con 2MB Flash.
- Puerto micro-USB para alimentación y datos.
- Dispositivo de 40 pines.
- 26 puertos GPIO multifunciones:
    - 23 puertos GPIO exclusivamente digitales.

    - 3 puertos GPIO exclusivamente analógicos.

- Periféricos digitales:

    - 2x UART
    - 2x I2C
    - 2x SPI
    - 16x canales PWM

2.4. Aspectos técnicos

A continuación se presentan los aspectos técnicos más relevantes en la realización del presente trabajo.

2.4.1. Sistema de arquitectura abierta

En el presente trabajo, el concepto de arquitectura abierta se concentra en el desarrollo de sistemas con límites claros entre los componentes además de existir protocolos de interacción bien definidos [16].

Dado lo anterior, el objetivo de la arquitectura abierta es permitir una fácil actualización e instalación de nuevos componentes y funciones. El sistema debe eliminar las restricciones de propiedad y reemplazarlas con soporte para conectividad con diferentes dispositivos [17].

2.4.2. Modos de transmisión de datos

Los sistemas electrónicos de comunicaciones se pueden diseñar para manejar la transmisión sólo en una dirección, en ambas direcciones, sólo en una a la vez, o en ambas direcciones al mismo tiempo. A éstos se les llama modos de transmisión. Hay cuatro modos de transmisión posibles: simplex, semidúplex, dúplex y dúplex/dúplex [18].

En el presente trabajo se trata únicamente con los modos de transmisión semidúplex y dúplex, los cuales, se definen a continuación:

- Semidúplex (half duplex): Se puede realizar la transmisión en ambas direcciones, pero no al mismo tiempo. En este tipo de transmisión, por lo general se utiliza un único cable/hilo de datos. Este modo se usa en la respectiva comunicación con los servomotores Dynamixel Ax-12a.

- Dúplex (full duplex): Existe transmisión en ambas direcciones al mismo tiempo. Con respecto al modo anterior, se utilizan 2 cables/hilos de datos, de esta forma trabaja principalmente el microcontrolador Raspberry Pi Pico.

3. Metodología

En esta sección se presenta la metodología que se seguirá en el desarrollo del presente trabajo de tesis.

La Figura 4, muestra un diagrama de bloques de los pasos generales en la realización del controlador de movimientos para un robot humanoide.

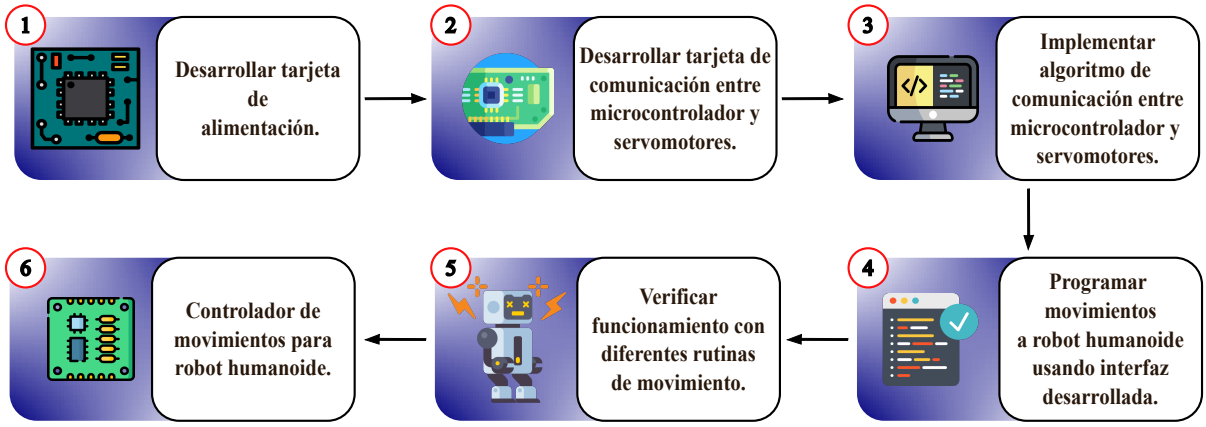


Figura 4: Diagrama de proceso general de la realización del proyecto.

1. **Desarrollo de tarjeta de alimentación:** Conocimiento previo sobre las respectivas magnitudes de voltaje de alimentación de los instrumentos y dispositivos a utilizar. Búsqueda de instrumentos reguladores de voltaje que proporcionen el voltaje requerido, además de la adquisición de dispositivos de protección.

2. **Desarrollo de tarjeta de comunicación:** Revisión de hojas de datos y manuales para conocer y verificar el lenguaje que utilizan los dispositivos y componentes a

utilizar. Búsqueda de dispositivos aptos en la conversión de modos de transmisión de datos, así como su correcta conexión e implementación.

3. **Implementación de algoritmo de comunicación:** Revisión de hojas de datos proporcionadas por fabricante para conocer la forma de envío y recepción de paquetes de datos para la correcta implementación y funcionamiento de los componentes que lo requieran. Aprendizaje de lenguaje de programación necesario en la programación de microcontrolador.

4. **Programación de movimientos:** Generación de programa de envío y recepción de paquetes de datos para los comandos requeridos por el sistema.

5. **Verificación de funcionamiento:** Implementación de diversas rutinas de movimiento variando posición y velocidad para verificar que el sistema trabaje de manera adecuada.

6. **Controlador de movimientos para robot humanoide:** Pruebas finales de diseño y funcionamiento para revisar la calidad de operación del controlador bajo periodos de tiempo cortos y prolongados.

4. Selección de instrumentación electrónica

En esta sección se describen los diferentes instrumentos a utilizar debido a las necesidades detectadas en el diseño y fabricación de un controlador de movimientos para un robot humanoide Bioloid Premium tipo A.

4.1. Transmisión de datos

El microcontrolador a utilizar en el desarrollo de este proyecto, es el dispositivo Raspberry Pi Pico y, los motores que componen al robot humanoide, corresponden a los motores Dynamixel Ax-12a. En la Sección 2.4.2, se observa que la forma de transmisión de datos para ambos dispositivos es Dúplex (full-duplex) y Semidúplex (half-duplex) respectivamente.

Debido a lo anterior, la forma de transmisión de datos en ambos dispositivos es diferente y esto debe de adaptarse de manera que la transmisión sea la misma.

4.1.1. Buffer tri-estado

Una forma de solucionar el problema de la diferencia en los modos de transmisión de datos, existente entre el microcontrolador Raspberry Pi Pico y los motores Dynamixel Ax-12a, es aplicando el uso de las salidas lógicas tri-estados. La configuración tri-estado toma ventaja de la operación a altas velocidades de las configuraciones *pull-up/pull-down* y permite a las salidas ser conectadas por un único cable. Se le llama tri-estado debido a que otorga tres posibles estados de salida: Alto, Bajo y Alta Impedancia [19]. La Figura 5 ilustra un diagrama con las tres posibles salidas.

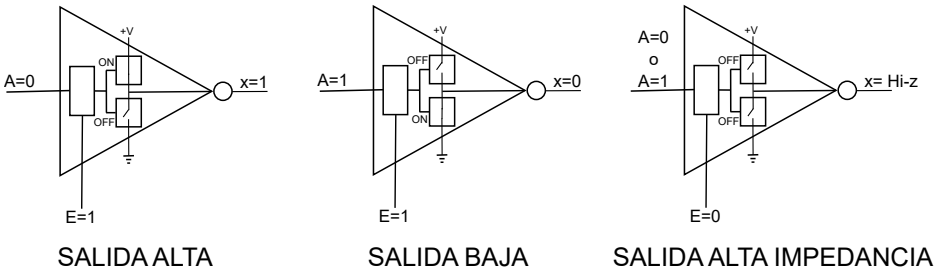


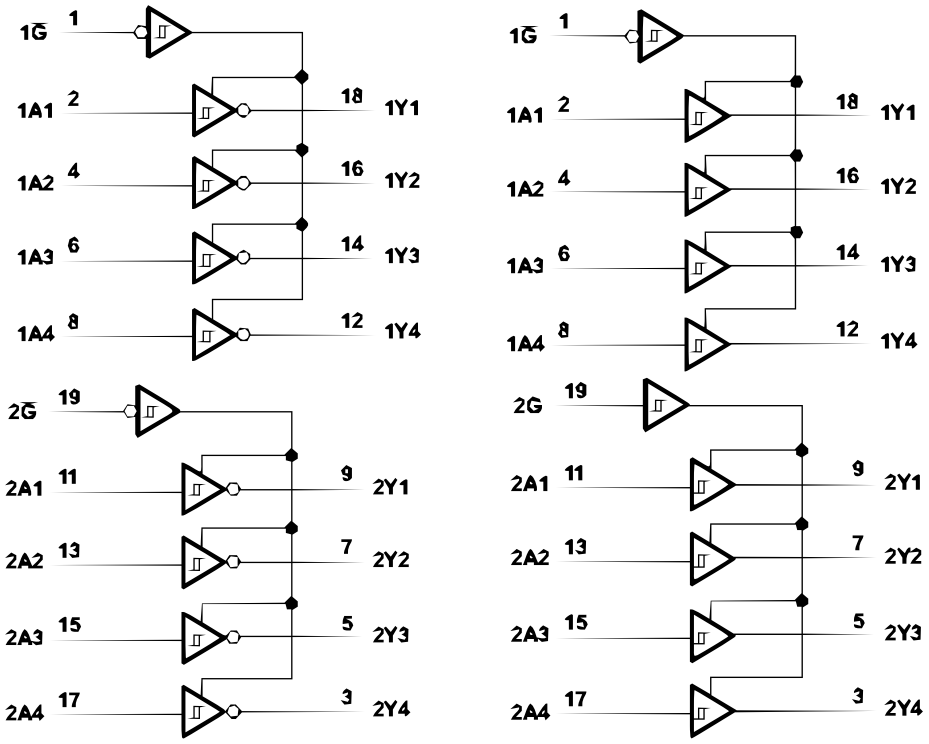
Figura 5: Tres condiciones de salida de tri-estado.

Aplicando la solución anterior, se encuentra un dispositivo tri-estado capaz de proporcionar una correcta conversión de transmisión de datos para poder comunicar el microcontrolador Raspberry Pi Pico y los motores Dynamixel Ax-12a.

El dispositivo que se encuentra pertenece a la familia SNx4LS24x, estos dispositivos comerciales ofrecen 8 buffers de salida tri-estado, dicha cantidad de buffers es ideal para el número de motores a controlar.

De la hoja de datos del fabricante de la familia de dispositivos anteriormente mencionados [20], se obtienen los diagramas lógicos de funcionamiento, véase las Figuras 6 y 7.

De igual manera, las tablas de funcionalidad de los dispositivos, se muestran en las Tablas 6, 7 y 8.



(a) SNx4LS240 (b) SNx4LS241

Figura 6: Diagramas lógicos.

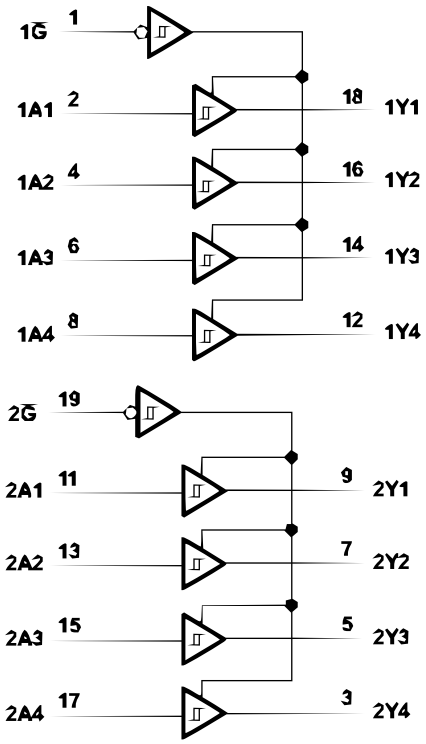


Figura 7: Diagrama lógico SNx4LS244.

Entradas		Salidas
$\bar{G}$	A	Y
L	L	H
L	H	L
H	X	Z

Tabla 6: Tabla de función de SNx4LS240.

Canal 1			Canal 2		
Entradas		Salidas	Entradas		Salidas
$1\bar{G}$	1A	1Y	2G	2A	2Y
L	L	L	H	L	L
L	H	H	H	H	H
H	X	Z	L	X	Z

Tabla 7: Tabla de función de SNx4LS241.

Entradas		Salidas
$\bar{G}$	A	Y
L	L	L
L	H	H
H	X	Z

Tabla 8: Tabla de función de SNx4LS244.

Donde, en los diagramas de las Figura 6 y 7, así como las Tablas 6, 7, 8, se tiene:

$$\bar{G} \; o \; G \; = \; \textit{Entrada de habilitador}$$

$$A \; = \; \textit{Señal de entrada}$$

$$Y \; = \; \textit{Señal de salida}$$

Las Tablas 6, 7 y 8, de funcionalidad, se siguen dependiendo del tipo de buffer tri-estado conseguido, de manera que esta tabla delimite las conexiones a realizar debido a los diversos funcionamientos del habilitador $\bar{G}$ o G.

4.1.2. Convertidor de nivel lógico

El microcontrolador Raspberry Pi Pico tiene un voltaje de trabajo de 3.3v (véase hoja de datos [21]). La familia de dispositivos de buffer tri-estado mencionados anteriormente, trabajan con un voltaje de operación de 5v como se puede revisar en la hoja de datos [20], si bien, dentro de sus características se especifica que el voltaje típico en las salidas es de 3.4v lo ideal es, no tomar este valor como base para trabajar ya que, este voltaje es obtenido bajo condiciones muy específicas y el mismo puede variar.

Prestando atención a los voltajes de funcionalidad de los dos dispositivos mencionados, claramente estos son diferentes y, tal como se especifica en sus respectivas hojas de datos, trabajar en voltajes superiores a los indicados puede ocasionar que la vida útil de los dispositivos disminuya o, en casos más graves, que estos dejen de funcionar.

Una aplicación entre ambos dispositivos es posible siempre y cuando se respeten los niveles de voltaje entre ellos. La conectividad existente de la Raspberry Pi Pico en dirección al dispositivo de tri-estado es posible y funcional, sin embargo, el enlace del dispositivo tri-estado al microcontrolador no es adecuado debido que el voltaje recibido por el microcontrolador es ligeramente superior al admitido, esto puede ocasionar que la vida útil se vea disminuida considerablemente.

Una forma de solucionar este problema de niveles de tensión es aplicando el uso de convertidores de nivel lógico o *level shifters* por su nombre en ingles.

Los convertidores de nivel son usados para convertir la señal lógica desde un nivel de voltaje a otro nivel. Son componentes importantes del circuito en sistemas de múltiples voltajes y se utilizan entre los circuitos centrales y el circuito de I/O [22]. Los convertidores de nivel proveen comunicación entre dos diferentes dominios de voltaje sin incluir ningún pin de suministro adicional [23].

4.2. Regulación de alimentación

Como se ha visto a lo largo del proyecto, el conjunto de dispositivos que componen todo el sistema funcionan a niveles de tensión y corrientes diferentes, sin embargo, lo ideal es que únicamente se utilice un solo suministro de energía, esto sugiere entonces que dicho suministro tenga una respectiva regulación que sea capaz de distribuir de manera equitativa la alimentación entre todos los componentes.

Existen de manera comercial circuitos reguladores de voltaje con principios de funcionamiento y niveles de eficiencia distintos, una buena pregunta es ¿cual utilizar para la regulación de voltaje?

Los reguladores en serie o lineales son los dispositivos más comunes y comerciales, estos controlan la tensión de salida ajustando de forma continua la tensión de entrada. Tienden a ser muy ineficientes debido a la potencia consumida por el elemento en serie. Su eficiencia es alrededor del 20 % y solamente resultan eficaces para baja potencia [24]. Entre estos dispositivos se encuentran los reguladores típicos de tres terminales, la familia 78XX.

A causa de lo mencionado anteriormente, un regulador de este tipo no es conveniente para una aplicación como lo es el diseño de un controlador de movimientos para robot humanoide ya que este requiere que la distribución de alimentación sea idónea para tener un buen desempeño.

La opción propuesta en el presente trabajo de tesis es un regulador de tipo conmutación, o en otras palabras, un convertidor reductor, *buck converter* por su nombre en ingles. En los reguladores de conmutación, el elemento regulador es un transistor que está constantemente conmutando entre corte y saturación. En estas regiones de operación el transistor disipa muy poca potencia, contando con una eficiencia de alrededor del 80 %. [24].

5. Desarrollo de tarjeta de alimentación

En esta sección, se describen las etapas que se llevan a cabo para el diseño y fabricación de la tarjeta de alimentación para el controlador de movimiento, la cual, debe ser

capaz de alimentar los motores Dynamixel Ax-12a que componen al robot humanoide, al microcontrolador Raspberry Pi Pico y por último, cualquier dispositivo necesario en el diseño y fabricación del controlador de movimientos.

5.1. Magnitudes importantes

El primer paso en el diseño de la tarjeta de alimentación es conocer las magnitudes de voltaje y corriente necesarias para el suministro correcto a todos los dispositivos. Revisando las respectivas hojas de datos de los componentes a utilizar, la Tabla 9 presenta los componentes con sus respectivas magnitudes.

Componente	V_{cc}	I_{cc}
Raspberry Pi Pico	1.8 a 5.5v	$\geq 40mA$
Motor Ax-12a	9 a 12v	1.5A
Buffer tri-estado	4.75 a 5.25v	17 a 54mA

Tabla 9: Componentes a utilizar con magnitudes de suministro de alimentación.

La Tabla 9, presenta para todos los componentes un rango de consumo de corriente, esto es debido a diferentes factores:

- Raspberry Pi Pico: El consumo es debido a la cantidad de *hardware* externo que se utilice, entre más dispositivos, más corriente se demanda.
- Motor Ax-12a: El consumo es variable, el consumo máximo, el cual se muestra en la Tabla 9, es demandado cuando el motor se encuentra en su torque máximo.
- Buffer tri-estado: El consumo depende del estado en el que las salidas se encuentren, alto, bajo o deshabilitadas.

Conocidas las respectivas magnitudes de corriente y voltaje de los dispositivos a utilizar, se genera un diagrama esquemático con las conexiones respectivas (véase Figura 8).

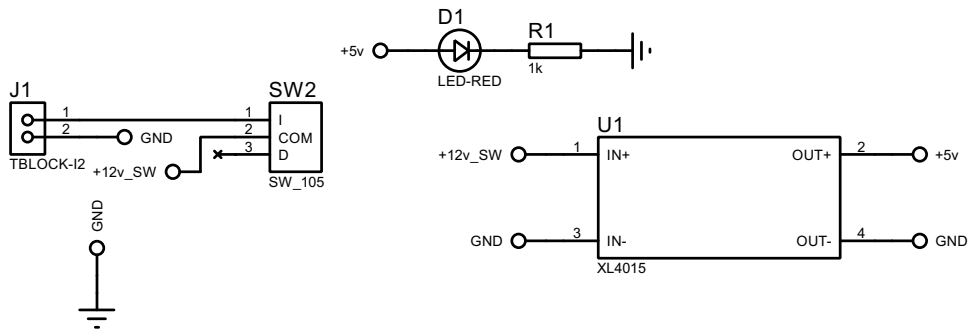


Figura 8: Diagrama esquemático de conexiones en tarjeta de alimentación.

De la Figura 8 existen puntos importantes a señalar:

- U1: El dispositivo con nombre XL4015 es un módulo de convertidor reductor o *buck converter*, este dispositivo proporciona una salida ajustable de 1.25v - 32v a 5A [25], esto lo hace un instrumento ideal para la regulación de voltaje.
- J1: Bloque de dos terminales en el cual se conecta la fuente de alimentación.
- SW2: Switch general para el corte y activación del flujo de energía para la tarjeta.
- D1: Foco led implementado con el objetivo de indicar que el sistema se encuentra encendido.

El diagrama esquemático que muestra la Figura 8, presenta dos nodos con niveles de tensión diferentes, el primero de 12v proveniente directamente de la fuente a utilizar, este nodo es el que alimenta a los motores Dynamixel Ax-12a, además, es el nodo de entrada del módulo reductor XL4015 y, el segundo nodo de 5v que sale del dispositivo regulador, este nodo es el que alimenta al resto de componentes. Se cuenta únicamente como dispositivo de protección un switch de corte general de energía, esto se debe a que, dentro de la etapa de alimentación, el dispositivo regulador de voltaje cuenta con protección contra corto circuito a la salida, por lo cual no es necesario agregar circuitería extra siempre y cuando la carga no sea excesiva. También los motores cuentan con protección, es por ello que su nodo de voltaje es directamente tomado de la fuente de alimentación, además, un punto importante a mencionar es que la colocación de algún fusible para la etapa de potencia, alimentación del conjunto de motores que componen al robot humanoide, es una tarea difícil de realizar desde el hecho que no es posible conocer el consumo de

corriente promedio de cada motor, debido a que, en cada motor este consumo es variable dependiendo del torque aplicado.

6. Desarrollo de tarjeta de comunicación

La tarjeta de comunicación comprende las conexiones necesarias para que exista comunicación bidireccional entre el microcontrolador Raspberry Pi Pico y los motores Ax-12a.

El manual de fabricante de los motores Dynamixel Ax-12a, presenta un diagrama electrónico general de comunicación entre los controladores de la marca propia Robotis (véase Figura 9), por lo tanto, con base en dicho diagrama y utilizando los componentes analizados en la sección 4.1, el diagrama esquemático diseñado con las respectivas conexiones se muestran en la Figura 10.

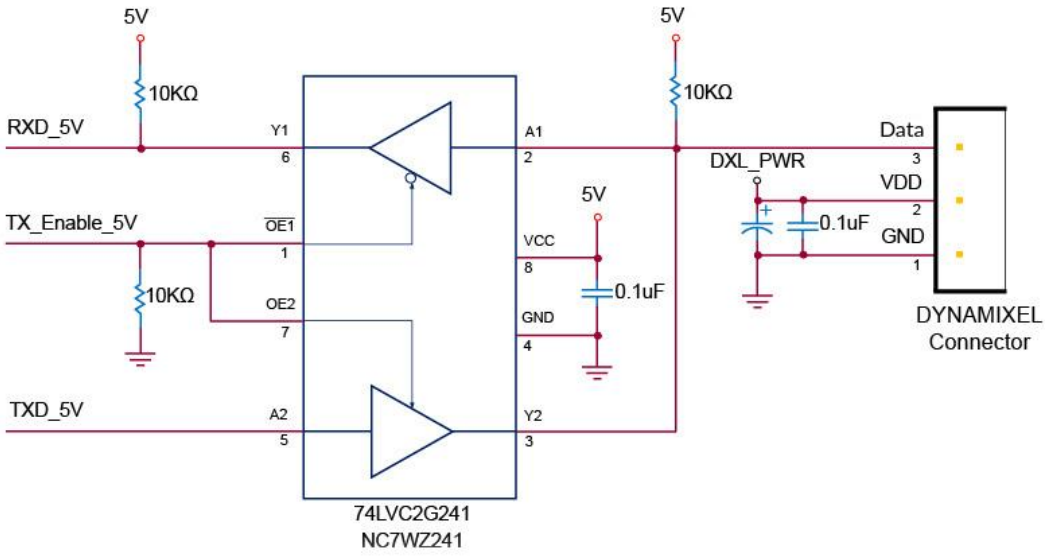


Figura 9: Diagrama electrónico general de comunicación, tomado del manual de fabricante Ax-12a.

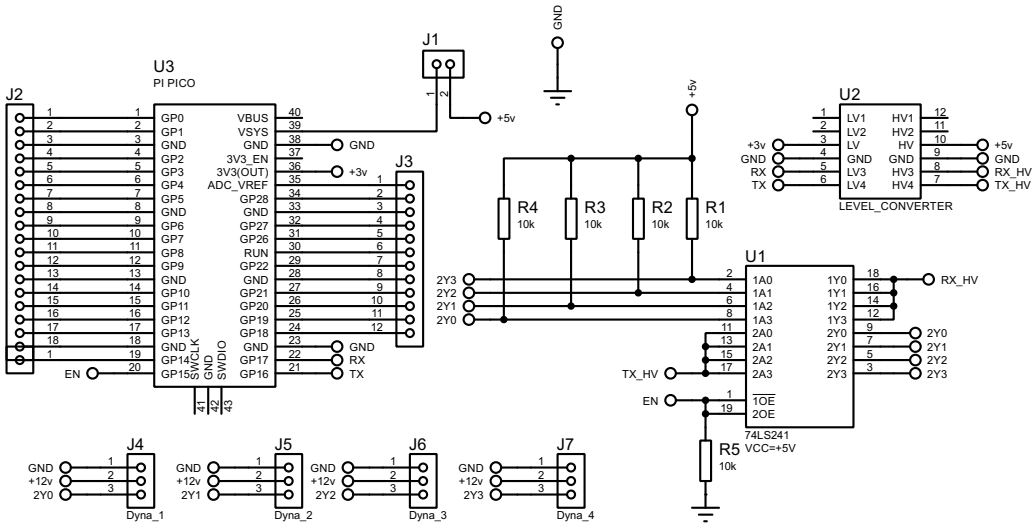


Figura 10: Diagrama esquemático de conexiones en tarjeta de comunicación.

A continuación, se desglosan los elementos mostrados en la Figura 10:

- U1: Circuito integrado 74LS241, este circuito corresponde al buffer tri-estado. La elección de este circuito es debida a que, la señal de habilitación es proporcionada por un solo pin de salida proveniente del microcontrolador.
- U2: Módulo de cuatro convertidores de nivel lógico (*level shifters*).
- U3: Microcontrolador de 40 pines Raspberry Pi Pico.
- J1: Puente tipo jumper utilizado para habilitar la alimentación externa del microcontrolador Raspberry Pi Pico, importante retirar este puente cuando se conecte el dispositivo por medio de cable micro USB.
- J2 y J3: Conectores tipo hembra colocados para la utilización de algún pin del microcontrolador.
- J4 a J7: Conectores tipo Molex utilizados para la conexión de los motores Dynamixel Ax-12a.

Estas conexiones se realizan en una tablilla de práctica, por su nombre en ingles *protoboard*, las cuales se muestran en la Figura 11.

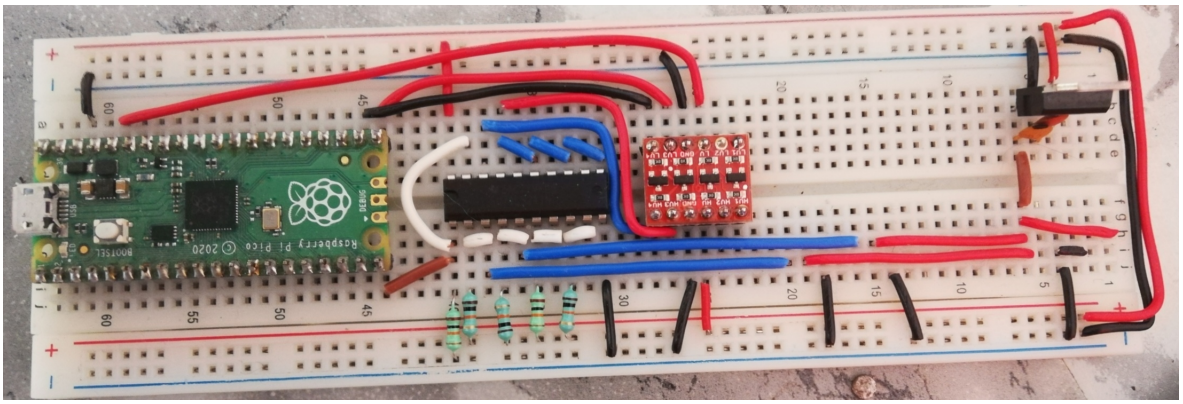


Figura 11: Circuito de prueba realizado en *protoboard*.

El circuito que presenta la Figura 11, se utiliza para ejecutar pruebas de comunicación y actuación del robot humanoide. Es sustancial mencionar que las conexiones expuestas en la Sección 5, no se llevan a cabo para las prácticas de funcionamiento, esto debido a que las ejecuciones de ensayo son realizadas en periodos de tiempo corto y bajo riguroso cuidado.

7. Algoritmo de comunicación

En esta sección se muestra el procedimiento para el desarrollo del algoritmo de programación para realizar una correcta comunicación entre el microcontrolador y los motores.

7.1. Transmisión de paquete de instrucción

En la Sección 2.2 se menciona que la comunicación efectiva dirigida a los motores Dynamixel Ax-12a consta de dos paquetes de datos, un primer paquete llamado paquete de instrucción y un segundo paquete, el paquete de estado. Ambos paquetes tienen su respectiva estructura, la cual, debe ser respetada para que la comunicación entre ambos dispositivos, microcontrolador y motores, sea la correcta.

El primer algoritmo a realizar es enfocado a la transmisión de instrucciones (véase Sección 2.2.2) para ello, el primer paso es el conocimiento de las características de comunicación de los motores, ya que, el microcontrolador es el dispositivo que se debe adaptar a los motores y no al contrario.

La Tabla 10, muestra la velocidad de comunicación en bps de los motores Dynamixel Ax12-a, se observa como es que existe un aumento de error en el movimiento de los motores conforme la disminución de la velocidad de comunicación.

Baud rate (bps)	Márgen de error
1M	0.000 %
500,000	0.000 %
400,000	0.000 %
250,000	0.000 %
200,000	0.000 %
115200	-2.124 %
57600	0.794 %
19200	-0.160 %
9600	-0.160 %

Tabla 10: Velocidad de comunicación de motores Dynamixel Ax-12a [3].

Considerando los datos presentados en la Tabla 10, el algoritmo de comunicación se trabaja en función de la velocidad más alta permitida, esto con el fin de evitar error en la ejecución de movimientos del robot.

La trama de datos de instrucción vista en la Sección 2.2.2, desglosa la cantidad de bytes a enviar dependiendo del tipo de instrucción que se deseé enviar, así como los respectivos parámetros. La Tabla 11 muestra las instrucciones que se pueden enviar a través del paquete de instrucción a los motores Ax-12a.

Valor	Instrucción	Descripción
0x01	Ping	Verifica si el paquete a llegado a un dispositivo con el mismo ID que el ID del paquete.
0x02	Read	Leer los datos del dispositivo.
0x03	Write	Escribir datos en el dispositivo.
0x04	Reg write	Registra el paquete en un estado de espera; El paquete se ejecuta más tarde a través de la instrucción Action.
0x05	Action	Ejecuta el paquete que se registró previamente usando Reg Write
0x06	Factory reset	Restablece la **tabla de control** a su configuración inicial predeterminada de fábrica.
0x83	Sync write	Para múltiples dispositivos, escribe datos en la misma dirección con la misma longitud a la vez.

Tabla 11: Instrucciones permitidas en motores Dynamixel Ax-12a.

Después de conocer el tipo de instrucciones que pueden enviarse a los motores, lo siguiente es saber los parámetros de cada instrucción. El manual de fabricante de los motores Dynamixel Ax-12a, proporciona la **tabla de control** referente a los parámetros de lectura y escritura [3]. El presente proyecto se limita únicamente a la lectura de posición presente en los motores y a la ejecución de movimiento, sin embargo, es importante aclarar que el método descrito en esta sección es funcional para cualquier otro parámetro tomado de la **tabla de control**.

Nombre de dato	Dirección	Tamaño (byte)	Acceso
Torque On/Off	0x18	1	RW
Posición objetivo	0x1E	2	RW
Posición presente	0x24	2	R

Tabla 12: Tabla de control simplificada.

La Tabla 12 presenta algunos de los parámetros de la tabla de control. Las letras R y

W de la columna acceso refieren a lectura y escritura respectivamente.

7.1.1. Ejecución de movimiento

El interés principal del presente proyecto, alusivo al manejo de datos, es la transmisión de instrucción referente al movimiento del robot. Por lo que el algoritmo de comunicación desarrollado debe cumplir con la estructura adecuada, además, de la precisión de los valores para cada byte contenido dentro del paquete de instrucción. La Figura 12 presenta un diagrama de flujo con respecto al procedimiento a seguir para la generación de código para la transmisión de instrucción.

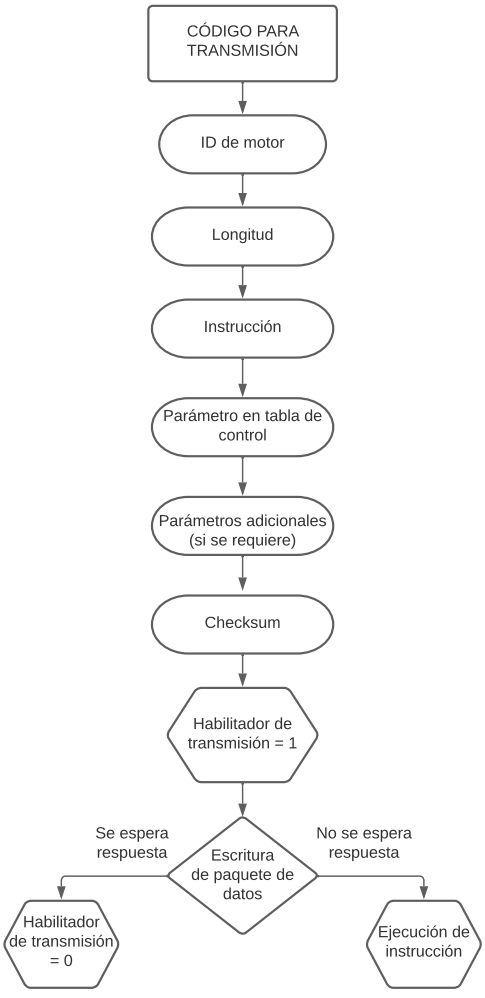


Figura 12: Diagrama de flujo para la transmisión de paquete de instrucción.

Siguiendo el diagrama de flujo presentado en la Figura 12, es posible generar un código inicial de prueba para observar que el paquete de datos es generado correctamente.

Para generar un paquete de datos de instrucción para el movimiento de un motor, el manual del fabricante menciona que se requieren los parámetros del valor de la posición objetivo y la velocidad de movimiento. En la Figura 13, se muestra el ángulo de operación del motor Dynamixel Ax-12a.

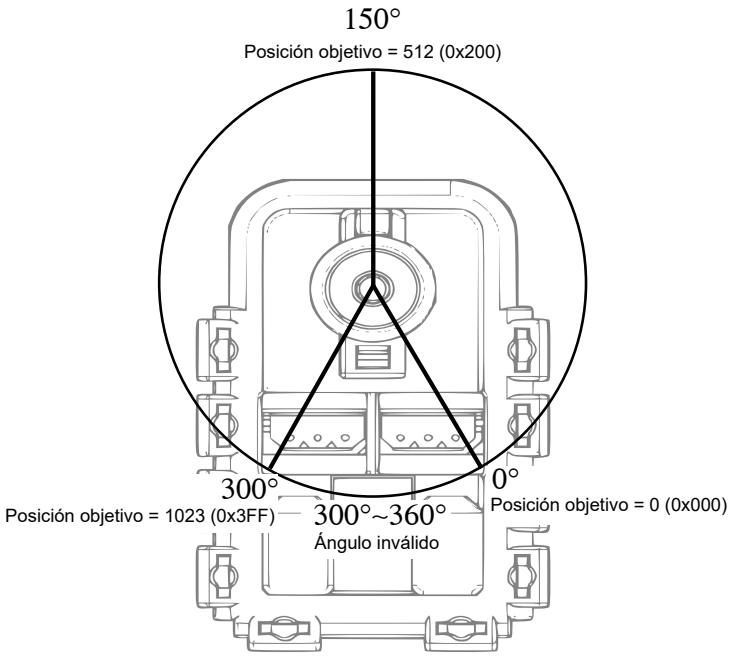


Figura 13: Región de movimiento del motor Dynamixel Ax-12a.

Como se observa en la Figura 13, el tamaño máximo del dato de la posición de un motor es de diez bits (1023 = 0x3FF). Dentro de la trama de datos de un paquete de instrucción, el valor del ángulo de la posición objetivo se coloca, de izquierda a derecha, comenzando por el byte menos significativo y después los dos bits restantes.

Para la posición objetivo es posible hacer la relación mostrada en la ecuación (1).

$$P_{ob} = \frac{1023}{300°} P_o \tag{1}$$

Donde:

$$P_{ob} \; = \; Posicion\ objetivo\ en\ bits\ (sistema\ decimal)$$

$$P_o \; = \; Posicion\ objetivo\ en\ grados$$

La velocidad de movimiento, al igual que la posición objetivo, tiene un valor máximo de diez bits (1023 = 0x3FF), donde, el valor de cero representa la máxima velocidad del motor sin que exista un control de velocidad y el valor máximo (1023) representa una velocidad de alrededor 114 rpm. Por lo tanto, es posible aplicar la relación de la ecuación (2).

$$V_{mb} = \frac{1023}{114rpm} V_m \tag{2}$$

Donde:

$$V_{mb} \ = \ Velocidad\ del\ motor\ en\ bits\ (sistema\ decimal)$$
$$V_m \ = \ Velocidad\ del\ motor\ en\ rpm$$

Suponiendo que se requiere una velocidad de 30 rpm, aplicando la ecuación (2), se tiene que colocar en la trama de datos de instrucción el valor en bits de 269 ($0x10D$), respetando nuevamente la posición, comenzando con el byte menos significativo y después los 2 bits restantes.

A continuación, en la Figura 14 se presenta un código de prueba generado en lenguaje python, suponiendo que se solicita a un motor Dynamixel Ax-12a con ID = 1 que se mueva a la posición objetivo de 56 (aprox. 16,4°) a una velocidad de 30 (aprox. 3,3 rpm).

```
#POSICION OBJETIVO: 56[0x38] #VELOCIDAD: 30[0x1E]
checksum = 0
data = [0xFF, 0xFF, 0x01, 0x07, 0x03, 0x1E, 0x38, 0x00, 0x1E, 0x00]
for j in range (2, len(data)):
    checksum = checksum + data[j]
checksum = ~checksum
checksum = checksum & 0x00FF
data.append(checksum)
print(data)
```

Figura 14: Código de prueba generado en python para comprobar el cálculo de checksum y la estructura del paquete de instrucción.

La Figura 14, muestra el código de prueba generado para la comprobación del cálculo de checksum, así como obtener una estructura de datos adecuada. A continuación, se explican algunas líneas del código:

- Línea 3: Inicialización de variable con nombre "data", contiene el arreglo de bytes referente a la trama de datos de instrucción, al final de este arreglo de bytes únicamente falta el byte correspondiente a checksum.
- Líneas 4 y 5: Ciclo *for* encargado de sumar los respectivos valores de los bytes contenidos en la variable "data", se omiten los primeros dos bytes.
- Línea 6: Instrucción que aplica el operador de negación a nivel bit.
- Línea 7: Instrucción que aplica el operador "and" a nivel bit, de manera que únicamente se conserve la información obtenida en los ocho bits menos significativos.
- Línea 8: Instrucción que añade al final del arreglo de bytes, contenido en la variable "data", el valor de checksum calculado.

El resultado obtenido al realizar la compilación del código de la Figura 14, se muestra en la Figura 15.

```
1 [255, 255, 1, 7, 3, 30, 56, 0, 30, 0, 128]
```

Figura 15: Resultado de código de prueba para cálculo de checksum y estructura de paquete de instrucción obtenido en entorno de desarrollo de lenguaje python.

Como se observa en la Figura 15, los valores obtenidos se presentan en sistema decimal, si estos datos son convertidos a sistema hexadecimal es posible darse cuenta que la trama de datos es adecuada.

Al comprobar que el código de prueba funciona, el siguiente paso es adecuar el código para la programación del microcontrolador Raspberry Pi Pico, utilizando lenguaje micropython, y de esta manera verificar la ejecución correcta de movimiento.

En la Figura 16, se muestra el código con el que se programa el microcontrolador Raspberry Pi Pico para la transmisión de instrucción de movimiento a un motor Dynamixel Ax-12a.

```
1  from machine import Pin, UART
2  import utime, time
3  uart0 = UART(0, baudrate = 1000000, tx = Pin(16), rx = Pin(17))
4  en1 = Pin(15, Pin.OUT)
5  def readpos(idq):
6      en1.value(1)
7      checksum = 0
8      data = [0xFF, 0xFF, idq, 0x07, 0x03, 0x1E, 0x38, 0x00, 0x1E, 0x00]
9      for j in range (2, len(data)):
10       checksum = checksum + data[j]
11     checksum = ~checksum
12     checksum = checksum & 0x00FF
13     data.append(checksum)
14     a = uart0.write(bytearray(data))
15     time.sleep(5)
16     en1.value(0)
17     time.sleep(5)
18 readpos(0x06)
```

Figura 16: Código de prueba en microcontrolador Raspberry Pi Pico para la ejecución de movimiento.

En la Figura 16, detallando las líneas de código más relevantes, se tiene:

- Líneas 1 y 2: Necesarias para el uso de GPIO del microcontrolador así como los módulos necesarios de tiempo para la incorporación de tiempo delay.
- Línea 3: Contiene la configuración de la respectiva comunicación serial dúplex, definiendo la velocidad de comunicación a 1Mbps.
- Línea 4: Define el pin del microcontrolador que funciona como habilitador para los grupos de buffers tri-estado; Es importante mencionar que este pin debe permanecer en estado alto cuando existe transmisión de información, tal como se muestra en la línea seis.

- Líneas 7 a 13: Se engloba el código de prueba mostrado en la Figura 14. Comprenden el paquete de datos de instrucción a enviar.
- Línea 14: Incluye el comando de escritura por medio de comunicación serial. El tipo de datos se especifica como un arreglo de bytes.

La ejecución de movimiento se muestra en la Figura 17, el movimiento es capaz de efectuarse siempre y cuando el paquete de datos se envíe de manera correcta.

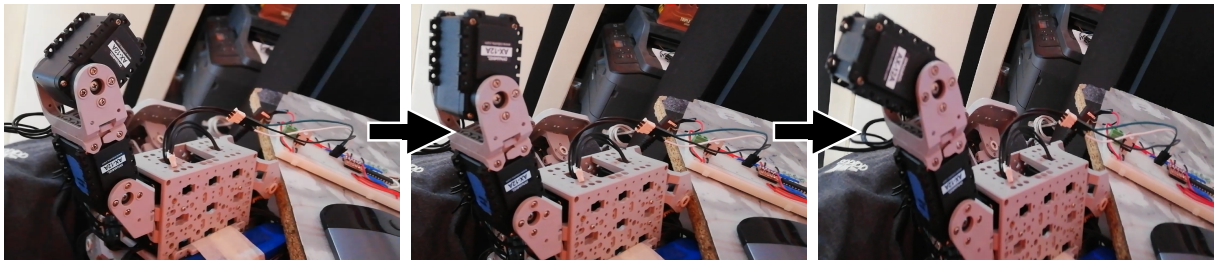


Figura 17: Movimiento efectivo de motor Dynamixel Ax-12a.

El movimiento efectuado y presentado en la Figura 17, corresponde a una instrucción de escritura con valor 0x03 (véase Tabla 11). Esta instrucción permite la ejecución de movimiento de un único motor a la vez, lo cual es eficiente si los movimientos a realizar son de manera secuencial y no consecutiva, sin embargo, el robot humanoide de manera ideal tiene que tener movimiento consecutivo para que el mismo sea fluido y natural.

El modo de conseguir un movimiento consecutivo es utilizando la instrucción de escritura *Sync write*. Esta forma de escritura sigue una forma similar de transmisión, sin embargo, algunos bytes de la trama de datos varían. Debido al hecho de que esta forma de ejecución de movimiento permite mover múltiples motores consecutivamente, el byte de ID se debe colocar en un valor de 254 (0xFE), lo que se pretende es que se realice un barrido de ID, dependiendo de la cantidad que se tenga, y no se establezca uno fijo, también, el byte de longitud cambia, ahora este valor debe ser calculado, el resto de la trama de datos es prácticamente igual con la diferencia de que, antes de ingresar los bytes de posición y velocidad, debe de colocarse el byte que indica el ID de cada motor. La ecuación (3) presenta el cálculo del byte de longitud.

$$l = ((L + 1) * N) + 4 \tag{3}$$

Donde:

$$L = Longitud\ de\ datos = 0x04$$

$$N = Numero\ de\ motores$$

Dadas las modificaciones anteriores, se genera el código de prueba que se presenta en la Figura 18, para mostrar que la trama de datos se forma de manera correcta y así evitar errores al momento que se desee enviar dicha trama a los motores.

```
n = 6; l = 4; bid = 0xFE; steps = 1; checksum = 0; speed = 0x1E
length = ((l+1)*n) + 4; pos = [[0x00, 0x00, 0x00, 0x323, 0x00, 0x128]]
for i in range(0,steps):
    data = [0xFF,0xFF, bid , length , 0x83 , 0x1E , l]
    checksum = 0
    for j in range (0,n):
        data.append(j+1)
        data.append(pos[i][j] & 0xFF)
        corr = (pos[i][j] >> 8)
        data.append(corr & 0x03)
        data.append(speed & 0xFF)
        corr = (speed >> 8)
        data.append(corr & 0x03)
    for j in range (2, len(data)):
        checksum = checksum + data [j]
    checksum = ~checksum
    checksum = checksum & 0x00FF
    data.append(checksum)
    pr = ""
    for t in range (0, len(data)):
        p = hex(data[t])
        pr = pr + p + " "
    print(pr)
```

Figura 18: Código de prueba para trama de datos de modo de escritura: *Sync write*.

La Figura 18, muestra un código más complejo debido a la cantidad de datos que se deben enviar en una única trama de datos. En este caso, se debe tener un arreglo de bytes para las diferentes posiciones objetivo de cada motor, si se observa la línea uno, la variable "n" representa la cantidad de motores a utilizar, es por ello que el arreglo de posiciones de la línea dos tiene un total de 6 bytes, representando las posiciones de cada motor. El resto de código es para colocar la trama de datos en el orden correcto. Es importante aclarar que este código funciona para ir barriendo los diferentes ID desde uno hasta el motor n-ésimo, si se desea enviar un arreglo de posiciones a motores aleatorios sin seguir el orden mencionado, se recomienda ingresar manualmente cada ID a utilizar.

Al compilar el código de la Figura 18, la trama de datos resultante se muestra en la Figura 19.

```
1 0xff/0xff/0xfe/0x22/0x83/0x1e/0x4/0x1/0x0/0x0/0x1e/0x0/0x2/0x0/0x0/0x1e/0
    x0/0x3/0x0/0x0/0x1e/0x0/0x4/0x23/0x3/0x1e/0x0/0x5/0x0/0x0/0x1e/0x0/0x6
    /0x28/0x1/0x1e/0x0/0x22/
```

Figura 19: Resultado de código de prueba para trama de datos de modo de escritura *Sync write* obtenido en entorno de desarrollo de lenguaje python.

Analizando la trama de datos de la Figura 19, es posible notar que se encuentra correctamente construida.

Adecuando el código de escritura *Sync write* para la programación del microcontrolador, el resultado en la ejecución del movimiento consecutivo de dos motores, se muestra en la Figura 20.

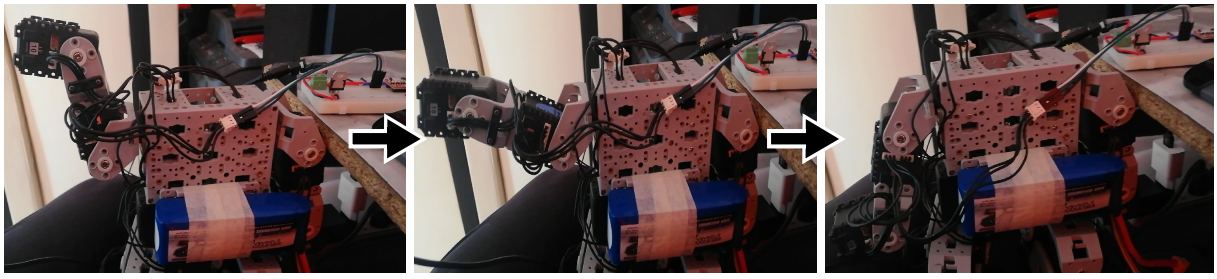


Figura 20: Movimiento efectivo de dos motores Dynamixel Ax-12a.

En la Figura 20, se observa que se realiza el movimiento consecutivo de dos motores,

aunque el código especifique el movimiento de seis, no hay problema si son conectados menos motores, siempre y cuando estos tenga un ID que se encuentre indicado dentro de la trama de datos.

7.2. Recepción de paquete de estado

Hasta ahora se a revisado la transmisión de un paquete de instrucción, toca turno a la recepción de un paquete de estado proveniente de los motores. La forma en como se recibe es primeramente enviando un paquete de datos de instrucción, si dicha instrucción le solicita al motor que envíe algún dato de interés o en todo caso se presenta algún error, entonces se recibe un paquete de estado, de cualquier otra forma no es posible recibir un paquete de estado ya que el motor tiene un funcionamiento único y exclusivo de esclavo.

La Figura 21, presenta un diagrama de flujo ilustrando la forma en como se debe recibir un paquete de estado proveniente de un motor.

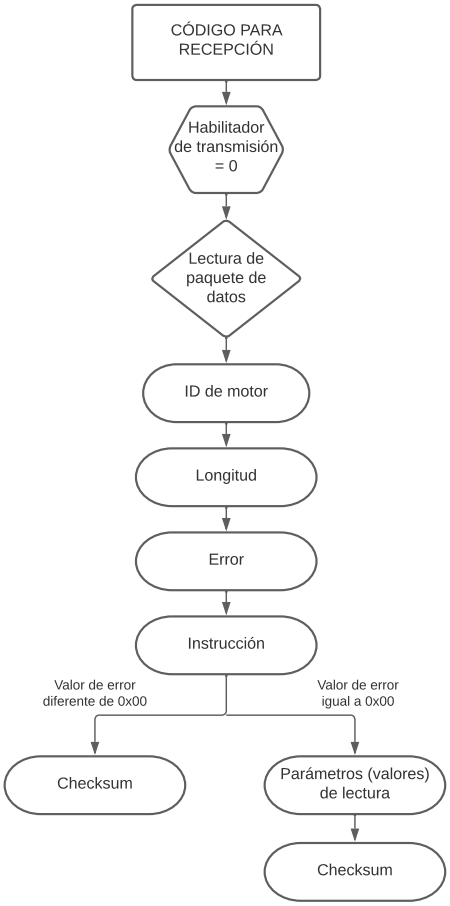


Figura 21: Diagrama de flujo para la recepción de paquete de estado.

En el diagrama de flujo de la Figura 21, es de relevancia prestar atención al byte que corresponde al error, si este byte presenta cualquier valor diferente de cero es debido a que existe algún error y al motor no le es posible retro alimentar con la información y/o ejecutar la instrucción solicitada. Para conocer cual es el error presente debe de revisarse la Tabla 13.

Bit	Error	Descripción
Bit 6	Instrucción	La instrucción enviada no existe o se envía instrucción de acción sin enviar un *Reg write.*
Bit 5	Sobrecarga	La carga actual no puede ser controlada por el torque establecido.
Bit 4	Checksum	El checksum del paquete de instrucción es incorrecto.
Bit 3	Rango	La instrucción está fuera del rango de uso.
Bit 2	Sobrecalentamiento	La temperatura interna del motor está fuera del rango de temperatura de funcionamiento establecido en la tabla de control.
Bit 1	Ángulo límite	La posición objetivo se escribe fuera del rango de límite de ángulo.
Bit 0	Voltaje de entrada	El voltaje aplicado está fuera del rango de voltaje operativo establecido en la tabla de control.

Tabla 13: Errores posibles exhibidos en paquete de estado.

Como se menciona anteriormente, para visualizar la recepción de un paquete de estado se debe primeramente enviar un paquete de instrucción proveniente del dispositivo maestro, para este caso, el dispositivo maestro es el microcontrolador Raspberry Pi Pico.

Siguiendo el diagrama de flujo referente a la transmisión de paquete de datos de instrucción y mostrado en la Figura 12, se solicita la lectura de la posición de un motor con ID igual a uno, el código implementado y cargado al microcontrolador se muestra en la Figura 22.

```
from machine import Pin, UART
import utime, time
uart0 = UART(0, baudrate = 1000000, tx = Pin(16), rx = Pin(17))
en1 = Pin(15, Pin.OUT)

rx1 = bytes()
def readpos(idq):
    en1.value(1)
    checksum = 0
    data = [0xFF, 0xFF, idq, 0x04, 0x02, 0x24, 0x02] #Leer posicion
    for j in range (2, len(data)):
      checksum = checksum + data[j]
    checksum = ~checksum
    checksum = checksum & 0x00FF
    data.append(checksum)
    a = uart0.write(bytearray(data))
    if a == len(data):
        utime.sleep_us(1)
        en1.value(0)
    rxdata = bytes()
    while uart0.any() > 0:
        rxdata += uart0.read()
    time.sleep(0.1)
    return(rxdata)

rx1 = readpos(0x01)
pr = ""
for t in range (0, len(rx1)):
    p = hex(rx1[t])
    pr = pr + p + " "
print(pr)
print(rx1)
```

Figura 22: Código implementado en microcontrolador para la lectura de posición de un motor.

A continuación, se presenta la explicación de algunas de las líneas del código mostrado en la Figura 22:

- Línea 8: Se define el pin del microcontrolador, que funge como habilitador del buffer tri-estado, para tener una salida en estado alto, se realiza previo a la escritura del paquete de instrucción.
- Línea 9 a 15: Estructuran la respectiva trama de datos de instrucción así como el cálculo de checksum.
- Línea 16: Se escribe por medio del hilo de transmisión el paquete de instrucción.
- Líneas 17 y 18: Condicional que solicita la longitud del dato que se escribe anteriormente, además se agrega un tiempo delay de 1 microsegundo para asegurar que la escritura se realiza completamente antes de apagar el habilitador del buffer tri-estado.
- Línea 19: Se define con estado bajo el pin del microcontrolador que desempeña el cargo de habilitador. Esta indicación es obligatoria y esencial antes de recibir información.
- Línea 21 y 22: Se inicializa un bucle *while* indicando que, si el canal de recepción recibe datos mayores a 0 la información se almacena en la variable *rxdata*.
- Líneas 28 a 30: Líneas que convierten la información que se recibe como un arreglo de bytes en una cadena de caracteres, esto se realiza para tener una mejor visualización de la información.

Después de emitir el código expuesto en la Figura 22, el resultado obtenido en el compilador con la información proveniente del motor, se muestra en la Figura 23.

```
1 b'\xff\xff\x01\x04\x00\xc3\x007'
2 0xff 0xff 0x1 0x4 0x0 0xc3 0x0 0x37
```

Figura 23: Paquete de estado recibido por el microcontrolador proveniente del motor Dynamixel Ax-12a.

La Figura 23, presenta la respuesta del motor con ID de uno, la primer línea muestra el resultado en un arreglo de bytes y la segunda línea lo muestra como una cadena de caracteres, si se observan ambos resultados detalladamente, estos, no son iguales debido que la respuesta expresada como un arreglo de bytes en ocasiones no muestra la información en sistema hexadecimal, sino como carácter (debido al valor ASCII), por este motivo es que se realiza la conversión a cadena de caracteres con la finalidad de tener una apreciación más clara de la respuesta. Debido a lo anterior y enfocando la atención en el resultado de la línea dos, se tiene primeramente que no existe error, después se muestra que la posición actual del motor tiene un valor de 0xC3, o de 195 en sistema decimal y, recordando la ecuación (1), este valor equivale a 57,184°.

Dada la respuesta obtenida, el procedimiento presentado en el diagrama de flujo de la Figura 21, funciona cuando se requiere la recepción de un paquete de estado de cualquier motor Dynamixel Ax-12a.

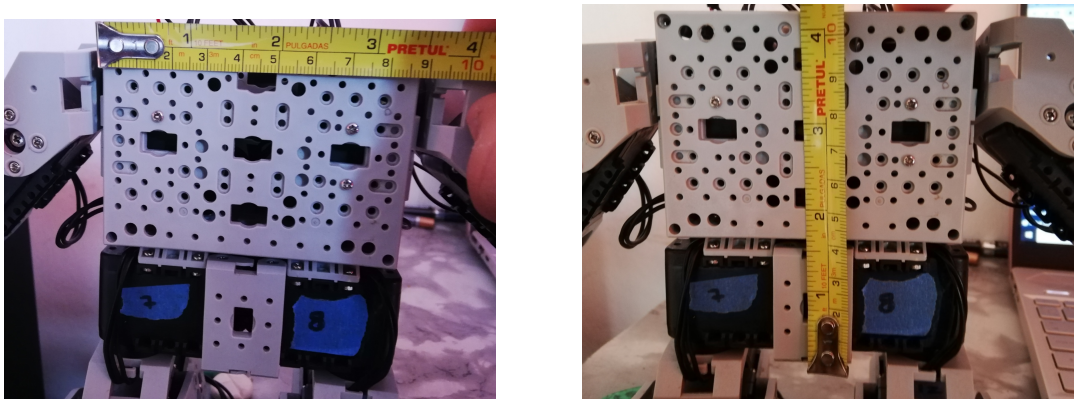
8. Resultados

En esta sección se presentan los resultados particulares obtenidos en el desarrollo del presente trabajo de tesis.

8.1. Unificación de tarjeta de alimentación y comunicación

En secciones anteriores se muestran los diagramas esquemáticos que componen la etapa de alimentación y de comunicación del sistema, además se realizan pruebas de funcionamiento con el objetivo de corroborar que dichas conexiones funcionan de manera adecuada.

La intención del diseño y fabricación de un controlador de movimientos para un robot humanoide, en este caso el robot Bioloid Premium tipo A, es que dicho controlador se coloque dentro del robot y este tenga un tamaño y forma adecuado que permita que el robot tenga movimiento libre y fluido. Para lograr lo anterior, es de relevancia que se conozca el espacio permitido dentro del robot. La Figura 24 muestra las respectivas medidas del torso del robot humanoide.



(a) Base (b) Altura

Figura 24: Medidas del torso de robot humanoide Bioloid Premium tipo A.

De la Figura 24, se aprecia que el torso del robot humanoide tiene un espacio libre de 88 x 107 mm, este espacio se encuentra disponible para la colocación de cualquier dispositivo debido que en esta superficie no se tiene ningún tipo de articulación y las extremidades del robot no colisionan con el área del torso.

La Figura 25 presenta el acomodo de componentes y las pistas que unen a los mismos, de manera que estas representen la base para la fabricación de una tarjeta PCB.

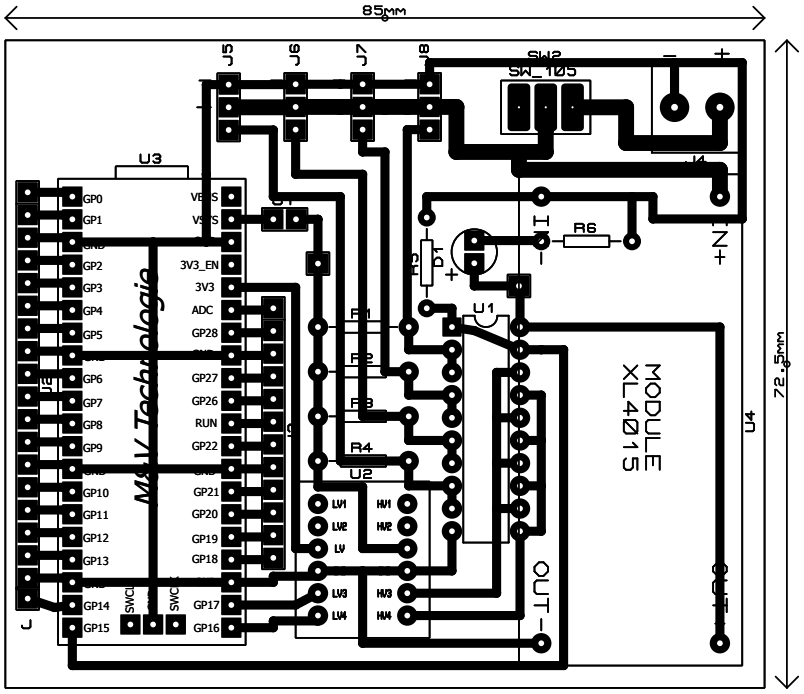


Figura 25: Dibujo de pistas de placa PCB.

Como se observa en la Figura 25, la forma y el tamaño de la respectiva tarjeta es un rectángulo de 85 x 72.5 mm, estas dimensiones se encuentran dentro del espacio disponible por el robot humanoide, esto convierte el diseño en una opción para su fabricación y montado sobre el torso del robot.

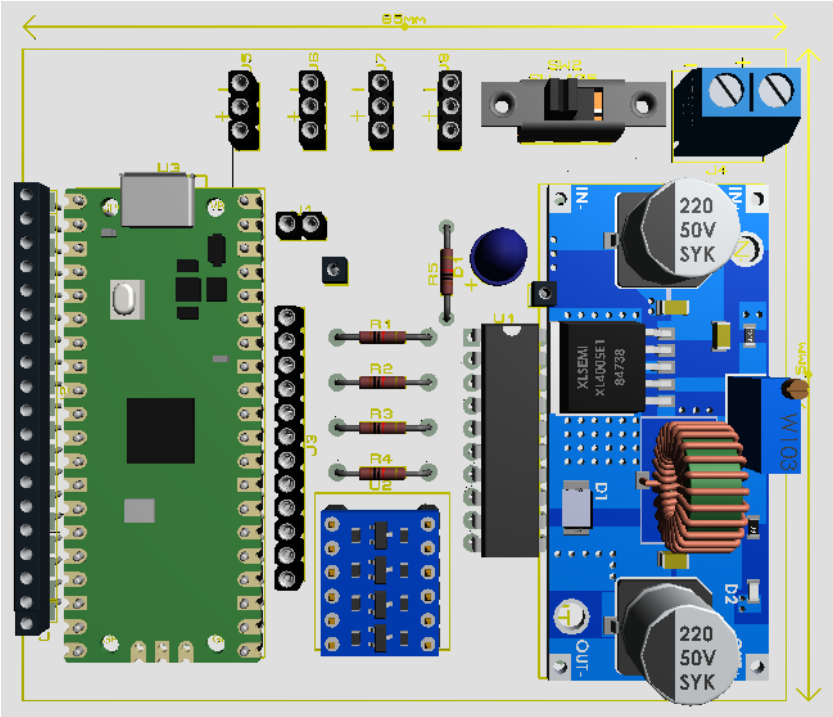
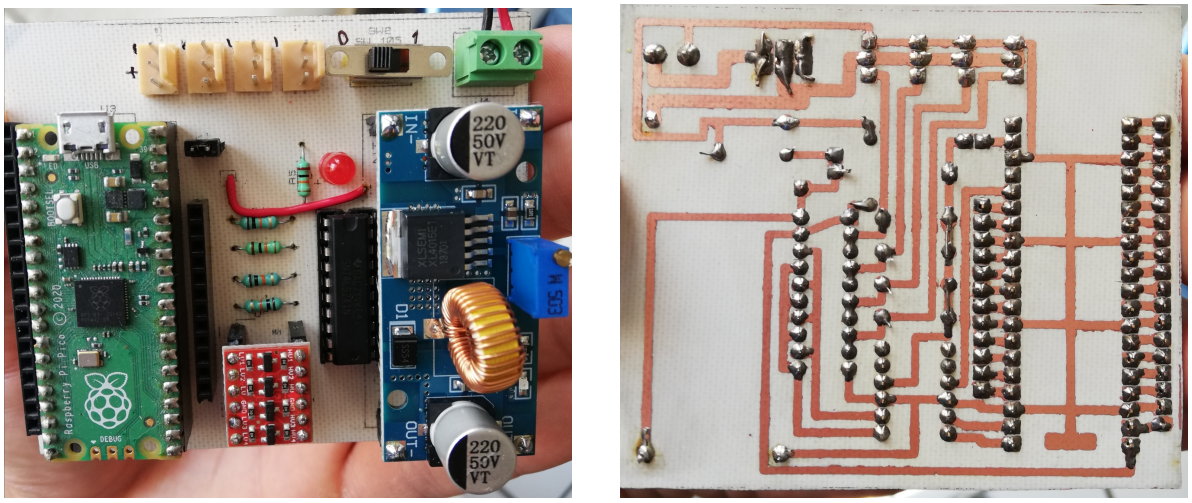


Figura 26: Modelado 3D de componentes en PCB.

La Figura 26, presenta un renderizado 3D de la tarjeta PCB, esta vista superior permite ver los componentes contemplados y la ubicación de los mismos cuando estos se coloquen en la respectiva tarjeta. Es de relevancia mencionar que el cuarteto de terminales triple tipo hembra expuestos en la parte superior de la Figura 26, no coinciden con los componentes a colocar de manera real, ya que en este espacio es donde se colocan los conectores tipo Molex de los motores.

La fabricación de la tarjeta PCB se realiza aplicando el método de “planchado”, la Figura 27, presenta el resultado final al colocar los componentes en su respectiva posición, de igual forma, también se enseñan las pistas de cobre que unen los instrumentos utilizados.



(a) Componentes (b) Pistas

Figura 27: Tarjeta PCB fabricada.

Al tener la tarjeta PCB que compone la tarjeta de alimentación y comunicación del controlador de movimientos, esta se coloca en el torso del robot humanoide, más específicamente en la parte posterior o espalda del robot, tal como se muestra en la Figura 28.

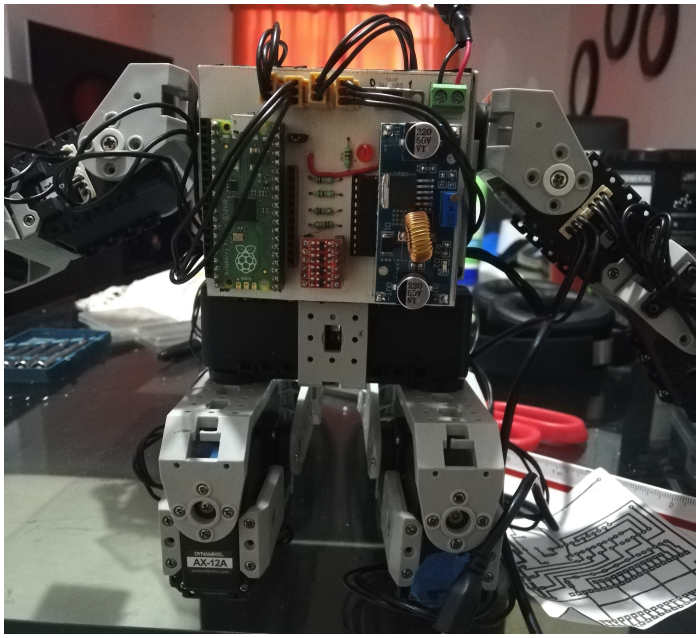


Figura 28: Colocación de tarjeta PCB en torso del robot humanoide.

La colocación en la espalda del robot se realiza de manera exitosa, dejando un espacio vacío y útil en la zona inferior del torso (véase Figura 28), superficie que es accesible para la colocación e implementación de una batería.

8.2. Controlador de movimientos para robot humanoide

Uno de los resultados más importantes es el diseño, fabricación y adecuación de la tarjeta de alimentación y comunicación, ahora toca turno al resto que conforma el producto final, un controlador de movimientos para robot humanoide, el cual es la programación.

Haciendo uso de la instrucción *Sync write*, siguiendo el procedimiento visto en la Sección 7.1.1, se da la orden al robot para realizar un caminado sencillo, dicha ejecución de movimiento se muestra en la Figura 29.

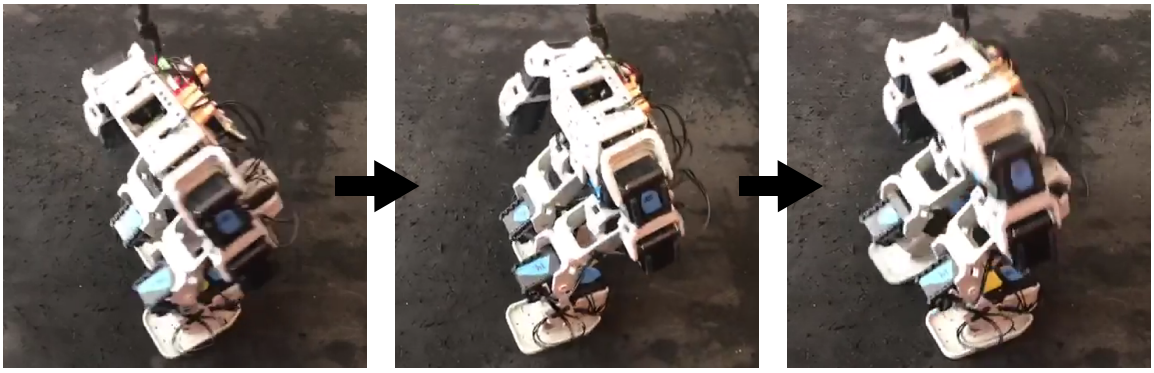


Figura 29: Ejecución de rutina de caminado de robot humanoide.

Como se muestra en la Figura 29, el controlador de movimientos se coloca en la espalda, y no existe alguna limitante u obstaculización ocasionada por el mismo, el caminar del robot se realiza de manera fluida. En el Anexo A, se muestra el código de programación aplicado.

Con la finalidad de corroborar que el dispositivo fabricado permite la interacción y la unión de otros instrumentos, se añade al sistema un módulo bluetooth, específicamente el módulo HC-05. Con la incorporación del módulo mencionado se permite el control del robot por medio de un dispositivo móvil.

La Figura 30 muestra el cometido de la integración del módulo bluetooth HC-05.

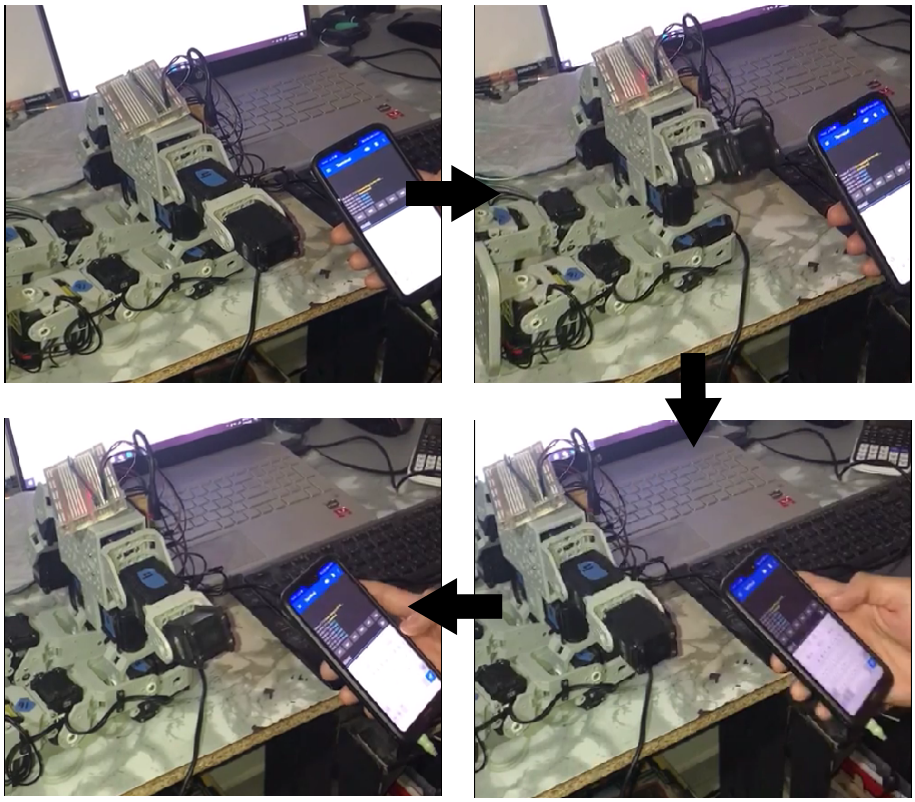


Figura 30: Ejecución de movimiento de robot integrando módulo bluetooth.

El modo en el que el robot ejecuta los respectivos movimientos es enviando una cantidad de cinco bytes de información por medio del dispositivo móvil, el primer byte indica el respectivo ID del motor que se desea que se mueva y el resto de bytes indican la posición; La velocidad de movimiento se define internamente en el código de programación.

Para la obtención de este resultado de integración de un dispositivo bluetooth, se utiliza la escritura de un paquete de instrucción de posición objetivo, de manera que el movimiento es individual para cada motor. En el Anexo B se presenta el código implementado así como las conexiones realizadas.

9. Conclusión

Con base en los resultados obtenidos en el diseño y fabricación de un controlador de movimientos para un robot humanoide, utilizando un microcontrolador Raspberry Pi Pico, se puede observar y concluir que el sistema construido ofrece múltiples ventajas en comparación a la competencia directa: el controlador de fábrica CM-530 con el que cuenta

de manera predeterminada el robot humanoide Bioloid Premium tipo A.

A continuación se listan algunas de las ventajas mencionadas anteriormente:

- Escalabilidad del sistema: El microcontrolador Raspberry Pi Pico implementado permite la incorporación de *hardware* externo sin importar fabricante, modo de comunicación, ni tipo de sistema, hablando entre digital y analógico, abriendo la posibilidad de añadir sensores, actuadores y/o otros microcontroladores o dispositivos.

- Modificación de *hardware*: Dado que el sistema es de arquitectura abierta, se abre la facultad de realizar actualizaciones o instalaciones de componentes y funcionalidades nuevas. Se sabe que conforme el tiempo pasa, la tecnología avanza y la existencia de microcontroladores de mayor gama aumenta, es por ello que el sistema desarrollado y fabricado presenta un medio para la actualización y el mejoramiento de *hardware*.

- Accesibilidad de programación: El microcontrolador Raspberry Pi Pico permite ser programado con dos lenguajes de programación diferentes, micropython y C++, por lo cual la accesibilidad de programación únicamente se limita al conocimiento del lenguaje a utilizar. No utiliza programación a bloques como lo permite de manera predeterminada el controlador CM-530, ni tampoco requiere cambios complejos en *firmware* para la liberación de método de programación.

- Sustitución de componentes: La instrumentación electrónica elegida y utilizada es comercial y de acceso en la región, esto permite el remplazo de componentes en caso de alguna falla o daño.

La Tabla 14, muestra de manera más clara una comparativa entre el controlador de movimientos desarrollado en el presente proyecto y el controlador de movimiento de fábrica CM-530.

Característica	CM-530	Controlador desarrollado
Peso	54g	52.87g
CPU	STM32F103RE	RP2040
V_{in}	6v - 15v	5v - 32v
GPIO	6 puertos 5-pin ROBOTIS; 5 conectores DYNAMIXEL series AX/MX	23 puertos digitales; 3 puertos analógicos; 2 x UART; 2 x I2C; 2 x SPI; 16 x canal PWM; 4 conectores Molex

Tabla 14: Comparación entre controlador de movimientos desarrollado y CM-530.

Fuera de las ventajas que presenta el controlador de movimientos diseñado y fabricado, es importante mencionar que, el dispositivo no solo tiene mejoras importantes en cuanto a escalabilidad se refiere, sino que también es completamente capaz de realizar las funciones que realiza el controlador de movimientos de fábrica CM-530 que trae consigo de manera predeterminada el robot humanoide Bioloid Premium tipo A, por lo que, el controlador de movimientos diseñado, fabricado y presentado en el presente trabajo de tesis, desde el punto de vista del autor, es una buena opción de relevo para el controlador de fábrica CM-530.

9.1. Trabajo Futuro

Más allá de los resultados obtenidos, algunas posibles direcciones para el trabajo futuro son:

- Rediseño de tarjeta PCB utilizando componentes con empaquetado SMD para minimizar tamaño de placa y tamaño de componentes.
- Creación de librerías o módulos en lenguaje mycropython para perfeccionar la programación referente al envío de paquetes de instrucción y la recepción de paquetes de estado.
- Diseño y construcción de interfaz gráfica de usuario compatible con lenguaje micropython para la visualización y tratamiento de datos provenientes de los motores.

- Diseño y construcción de aplicación para dispositivo móvil para el control y ejecución de movimientos del robot humanoide, así como obtención de datos provenientes del robot.

Referencias

[1] *BIOLOID Premium*, ROBOTIS, 2022, consulta [10/08/2022]. [Online]. https://emanual.robotis.com/docs/en/edu/bioloid/premium/

[2] *CM-530*, ROBOTIS, 2022, consulta [07/08/2022]. [Online]. https://emanual.robotis.com/docs/en/parts/controller/cm-530/

[3] *Dynamixel AX-12*, ROBOTIS, 2006.

[4] C. C. Carman, "El mecanismo de anticitera.: Una computadora astronómica de la antigüedad," *Ciencia hoy*, vol. 21, no. 123, pp. 32–38, 2011.

[5] S. Kumar Saha, *Introducción a la robótica.* México editorial McGraw. Hill, 2010.

[6] Real Academia Española , *Robot*, 23rd ed., 2021, consulta [07/08/2022]. [Online]. https://dle.rae.es/robot

[7] H. P. Moravec, *Robot*, 2022, consulta [07/08/2022]. [Online]. https://www.britannica.com/technology/robot-technology.

[8] G. X. Ortiz Morales, J. N. Garrido Vázquez, A. Hernández Cadena, J. A. Magaña, J. M. Gómez Zea, and D. M. de la O, "Acercamiento a la robótica: Robot humanoide," *Innovación y desarrollo tecnológico revista digital*, vol. 12, no. 4, 2020.

[9] C. Burghart, R. Mikut, R. Stiefelhagen, T. Asfour, H. Holzapfel, P. Steinhaus, and R. Dillmann, "A cognitive architecture for a humanoid robot: a first approach," in *5th IEEE-RAS International Conference on Humanoid Robots, 2005.*, 2005, pp. 357–362.

[10] S.-J. Yi and D. D. Lee, "Dynamic heel-strike toe-off walking controller for full-size modular humanoid robots," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 395–400.

[11] FIRA worldcup, "About fira — fira roboworld cup official website," consulta [07/08/2022]. [Online]. https://firaworldcup.org/about/

[12] ROBOTIS, "ROBOTIS," 2022, consulta [20/09/2022]. [Online]. https://www.robotis.us/

[13] S. Behnke, M. Schreiber, J. Stuckler, R. Renner, and H. Strasdat, "See, walk, and kick: Humanoid robots start to play soccer," in *2006 6th IEEE-RAS International Conference on Humanoid Robots*, 2006, pp. 497–503.

[14] A. Zainuddin, M. Ahmed, M. M. Md Zan, and H. Hashim, "An alternative open architecture controller design for the bioloid humanoid robot," *Journal of Electrical & Electronic Systems Research*, vol. 16, pp. 1–9, 06 2020.

[15] G. Halfacree and B. Everard, *Get started with MicroPython on Raspberry Pi Pico*. Raspberry Pi Trading Ltd., 2021.

[16] S. Iarovyi, J. L. M. Lastra, R. Haber, and R. del Toro, "From artificial cognitive systems and open architectures to cognitive manufacturing systems," in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 2015, pp. 1225–1232.

[17] A. Zainuddin, B. Ali, M. M. M. Zan, R. Hashim, and H. Hashim, "An open-architecture humanoid robot controller in support of developmental disability (dd) rehabilitation," in *2017 International Conference on Electrical, Electronics and System Engineering (ICEESE)*, 2017, pp. 42–47.

[18] W. Tomasi, *Sistemas de comunicaciones electrónicas*. Pearson educación, 2003.

[19] N. S. Widmer, R. J. Tocci *et al.*, *Digital systems: Principles and applications*. Prentice Hall. Pearson Education International, 2007.

[20] *SNx4LS24x, SNx4S24x Octal Buffers and Line Drivers With 3-State Outputs*, Texas Instruments, 10 2016.

[21] *Raspberry Pi Pico Datasheet An RP2040-based microcontroller board*, Raspberry Pi Ltd, 2020.

[22] M. Kumar, S. K. Arya, and S. Pandey, "Level shifter design for low power applications," *arXiv preprint arXiv:1011.0507*, 2010.

[23] T.-H. Chen, J. Chen, and L. Clark, "Subthreshold to above threshold level shifter design," *J. Low Power Electronics*, vol. 2, pp. 251–258, 08 2006.

[24] G. A. R. Robredo, *Electrónica básica para ingenieros.* Ed. Universidad de Cantabria, 2014, vol. 45.

[25] *5A 180KHz 36V Buck DC to DC Converter XL4015*, XLSEMI, rev 1.3.

Anexos

A. Código de rutina de caminado

En el Anexo A, se presenta el código referente a la rutina de caminado del robot humanoide, redactado en lenguaje de programación micropython, el cual es implementado en el microcontrolador Raspberry Pi Pico.

```
from machine import Pin, UART
import time, utime



uart0 = UART(0, baudrate = 1000000, tx = Pin(16), rx = Pin(17))
enb = Pin(15, Pin.OUT)
enb.value(1)



n = 18
l = 4
length = ((l+1)*n) + 4
bid = 0xFE
checksum = 0

steps = 28
pos = [[169, 722, 279, 744, 462, 561, 358, 666, 513, 522, 302, 660, 246,
   769, 621, 348, 513, 522],
          [166, 719, 279, 744, 462, 561, 358, 666, 518, 527, 308, 651,
             248, 761, 625, 348, 518, 527],
          [167, 720, 279, 744, 462, 561, 358, 666, 519, 533, 314, 650,
             252, 761, 628, 346, 523, 532],
          [171, 724, 279, 744, 462, 561, 358, 666, 513, 541, 321, 662,
             257, 778, 629, 341, 527, 537],
```

```
21            [179, 732, 279, 744, 462, 561, 358, 666, 504, 550, 327, 682,
                 262, 801, 630, 338, 530, 541],
22            [189, 742, 279, 744, 462, 561, 358, 666, 496, 556, 332, 704,
                 266, 820, 631, 342, 532, 543],
23            [201, 754, 279, 744, 462, 561, 358, 666, 493, 558, 335, 724,
                 267, 827, 633, 354, 533, 544],
24            [215, 768, 279, 744, 462, 561, 358, 666, 496, 556, 337, 736,
                 266, 820, 636, 374, 532, 543],
25            [230, 783, 279, 744, 462, 561, 358, 666, 504, 550, 338, 740,
                 262, 801, 641, 396, 530, 541],
26            [246, 799, 279, 744, 462, 561, 358, 666, 513, 541, 339, 737,
                 257, 778, 647, 416, 527, 537],
27            [260, 813, 279, 744, 462, 561, 358, 666, 519, 533, 340, 732,
                 252, 761, 654, 428, 523, 532],
28            [274, 827, 279, 744, 462, 561, 358, 666, 518, 527, 343, 730,
                 248, 761, 660, 427, 518, 527],
29            [285, 838, 279, 744, 462, 561, 358, 666, 513, 522, 347, 730,
                 246, 769, 666, 418, 513, 522],
30            [294, 847, 279, 744, 462, 561, 358, 666, 507, 516, 354, 726,
                 248, 775, 671, 409, 507, 516],
31            [300, 853, 279, 744, 462, 561, 358, 666, 501, 510, 363, 721,
                 254, 777, 675, 402, 501, 510],
32            [303, 856, 279, 744, 462, 561, 358, 666, 496, 505, 372, 715,
                 262, 775, 675, 398, 496, 505],
33            [302, 855, 279, 744, 462, 561, 358, 666, 490, 504, 373, 709,
                 262, 771, 677, 395, 491, 500],
34            [298, 851, 279, 744, 462, 561, 358, 666, 482, 510, 361, 702,
                 245, 766, 682, 394, 486, 496],
35            [290, 843, 279, 744, 462, 561, 358, 666, 473, 519, 341, 696,
                 222, 761, 685, 393, 482, 493],
36            [280, 833, 279, 744, 462, 561, 358, 666, 467, 527, 319, 691,
                 203, 757, 681, 392, 480, 491],
```

```
37            [268, 821, 279, 744, 462, 561, 358, 666, 465, 530, 299, 688,
                 196, 756, 669, 390, 479, 490],
38            [254, 807, 279, 744, 462, 561, 358, 666, 467, 527, 287, 686,
                 203, 757, 649, 387, 480, 491],
39            [239, 792, 279, 744, 462, 561, 358, 666, 473, 519, 283, 685,
                 222, 761, 627, 382, 482, 493],
40            [223, 776, 279, 744, 462, 561, 358, 666, 482, 510, 286, 684,
                 245, 766, 607, 376, 486, 496],
41            [209, 762, 279, 744, 462, 561, 358, 666, 490, 504, 291, 683,
                 262, 771, 595, 369, 491, 500],
42            [195, 748, 279, 744, 462, 561, 358, 666, 496, 505, 293, 680,
                 262, 775, 596, 363, 496, 505],
43            [184, 737, 279, 744, 462, 561, 358, 666, 501, 510, 293, 676,
                 254, 777, 605, 357, 501, 510],
44            [175, 728, 279, 744, 462, 561, 358, 666, 507, 516, 297, 669,
                 248, 775, 614, 352, 507, 516]]
45
46    vel = 100
47
48    while True:
49        for i in range(0,steps):
50            data = [0xFF,0xFF, bid , length , 0x83 , 0x1E , l]
51            checksum = 0
52            for j in range (0,n):
53                data.append(j+1)
54                data.append(pos[i][j] & 0xFF)
55                aux = (pos[i][j] >> 8)
56                data.append(aux & 0x03)
57                data.append(vel & 0xFF)
58                aux = (vel>> 8)
59                data.append(aux & 0x03)
60            for j in range (2, len(data)):
```

```
61              checksum = checksum + data [j]
62          checksum = ~checksum
63          checksum = checksum & 0x00FF
64          data.append(checksum)
65
66          uart0.write(bytearray(data))
67          utime.sleep_us(2)
68          time.sleep(0.05)
```

B. Implementación de módulo bluetooth

En el Anexo B, se muestra el diagrama de conexión y la programación referente al resultado de integración de un dispositivo bluetooth, expuesto en la Sección 8.2.

B.1. Diagrama de conexión

El diagrama de la Figura 31, presenta las conexiones realizadas del módulo bluetooth HC-05 con el controlador de movimientos diseñado. Como nota importante, para el resultado visualizado en la sección 8.2 el jumper de alimentación del microcontrolador J1 no se encuentra conectado, debido que el microcontrolador se encuentra alimentado por medio del puerto jack micro USB.

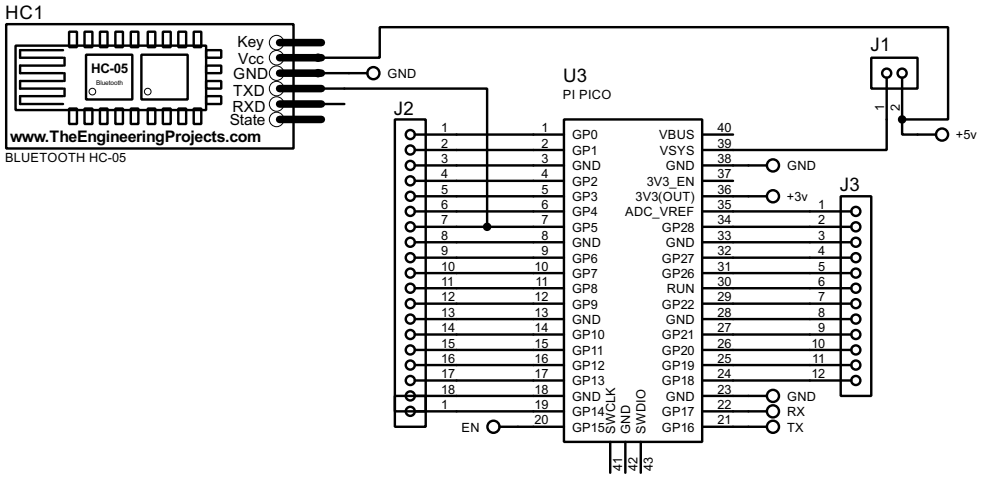


Figura 31: Conexión de módulo bluetooth a microcontrolador.

B.2. Código de programación cargado en microcontrolador

El Anexo B.2, presenta el código de programación, implementado en el microcontrolador Raspberry Pi Pico, para la ejecución de movimiento de los motores del robot humanoide, proveniente de los datos transmitidos por un dispositivo móvil y recibidos por el módulo bluetooth HC-05.

```
from machine import Pin, UART
import utime, time

uart0 = UART(0, baudrate = 1000000, tx = Pin(16), rx = Pin(17)) #motores
uart1 = UART(1, baudrate = 9600, tx = Pin(4), rx = Pin(5)) #bluetooth
en1 = Pin(15, Pin.OUT)

rxblue = bytes()

def writepos(idq,pos):
    en1.value(1)
    checksum = 0
```

```
13      vel = 0x1E
14      data = [0xFF, 0xFF, idq, 0x07, 0x03, 0x1E]
15      data.append(pos & 0xFF)
16      aux = (pos >> 8)
17      data.append(aux & 0x03)
18      data.append(vel & 0xFF)
19      aux = (vel >> 8)
20      data.append(aux & 0x03)
21      for j in range (2, len(data)):
22        checksum = checksum + data[j]
23      checksum = ~checksum
24      checksum = checksum & 0x00FF
25      data.append(checksum)
26      a = uart0.write(bytearray(data))
27      time.sleep(2)
28      en1.value(0)
29      time.sleep(2)
30
31  u = bytes()
32
33  while True:
34      while uart1.any() > 0:
35          rxblue += uart1.read()
36          q = rxblue[0:1]
37          e = rxblue[1:5]
38          w = int(q)
39          r = int(e)
40          writepos(w,r)
41          print(rxblue)
42          print(q)
43          rxblue = bytes()
```