UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Ingeniería de Sistemas y Automática

Universitat Politècnica de València

Proyecto final: Control de brazo robótico con 3 DOF mediante BeagleBone Black

Sistemas Empotrados

Máster Universitario en Automática e Informática Industrial

Autor: Juan Carlos Brenes Torres

Profesores: José Simó, Francisco Blanes

2015-2016

Índice de contenidos

Índice de figuras

Índice de tablas

1. Introducción

En el presente informe técnico se plasman los conocimientos aplicados, los retos encontrados y las soluciones implementadas para el proyecto desarrollado como trabajo final de la asignatura Sistemas Empotrados del Máster Universitario en Automática e Informática Industrial.

El objetivo de dicho proyecto consistió en controlar un brazo robótico de 3 grados de libertad (DOF, por sus siglas en inglés) e implementar sus ecuaciones de cinemática inversa mediante un sistema empotrado BeagleBone Black. En la Figura 1 se puede observar una fotografía del brazo robótico implementado.

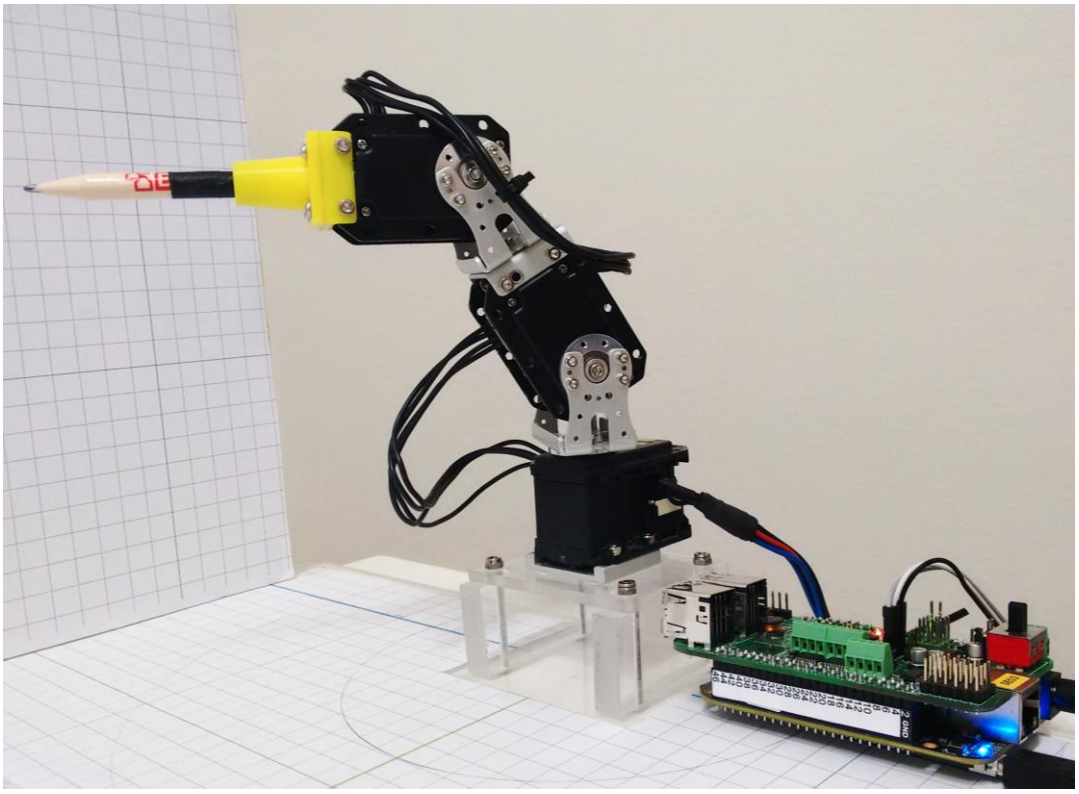

Figura 1. Brazo Robótico implementado y sistema BeagleBone Black

Para poder alcanzar el objetivo deseado en el proyecto y debido a la complejidad de los retos involucrados, donde se mezcla el control mecánico, la comunicación de bajo nivel y la creación de órdenes de más alto nivel; se planteó un modelo de desarrollo estructurado que se puede apreciar en la Figura 2.

Bajo este modelo primeramente se trabajó montando la estructura mecánica del robot: servomotores RX-10, plataforma para el brazo y el desarrollo del sujetador de herramienta. Luego de esto se trabajó con el control eléctrico mediante la tarjeta BeagleBone Black, específicamente se buscó el lograr comunicación serial mediante el protocolo RS-485.

Figura 2. Jerarquía de funciones desarrolladas en el proyecto

El siguiente paso consistió en configurar un mensaje de manera que fuera entendible por los servomotores y con esto poder enviar instrucciones, como lo es un mensaje para moverse a determinado ángulo. Habiendo logrado esto, se trabajó en implementar la cinemática inversa del brazo, esto con el fin de poder obtener la configuración de ángulos necesaria en los servos a partir de una coordenada deseada para que sea alcanzada por la punta del brazo.

Se implementó además dos generadores de trayectorias, uno de trayectorias lineales o cartesianas y otro de trayectoria libre. El objetivo con esto es poder especificar un punto inicial y uno final y que el brazo robótico describa dicha trayectoria.

Por último se desarrolló la capacidad de que la aplicación pudiera ejecutar cualquiera de las funciones desarrolladas. Por tanto dependiendo de los parámetros con que se llame la aplicación, se ejecutará un movimiento hacia la posición de reposo del robot, un movimiento hacia una configuración de ángulos dada, un movimiento hacia una coordenada dada, una trayectoria lineal entre dos puntos dados, una trayectoria libre entre dos puntos dados o una secuencia de demostración de los movimientos del robot.

En la sección de desarrollo del presente informe se procederá a explicar en detalle cada uno de estos niveles de jerarquía y las soluciones implementadas.

2. Desarrollo

2.1. Estructura mecánica

Como base del desarrollo del proyecto se tiene la estructura mecánica del brazo robótico. Esta estructura consiste inicialmente de una plataforma de plástico donde se montó el brazo y en donde además se añadió una cuadrícula. La función de esta cuadrícula es la ayudar a identificar los ejes cartesianos que se utilizan y ayudar a dimensionar puntos en el espacio. Para esta cuadrícula se escogió un espaciado de 1cm por división y se resaltó en color cada uno de los ejes. En la Figura 3 se observa el detalle de la plataforma utilizada junto con la cuadrícula.

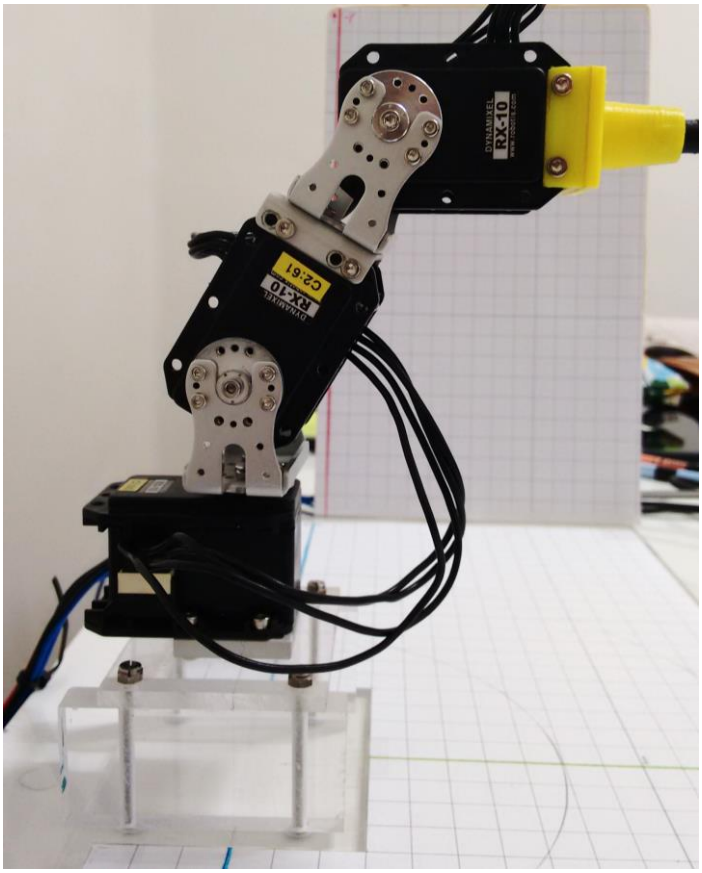

Figura 3. Estructura mecánica del brazo: plataforma, servomotores y sujetador de herramienta.

El brazo robótico implementado consiste básicamente de tres servomotores unidos entre sí mediante sujetadores de aluminio. Los servomotores usados son modelo RX-10 de la marca Dynamixel (ver Figura 4). Estos servomotores son dirigidos específicamente a aplicaciones de robótica y poseen características como comunicación digital (mediante protocolo RS-485), un rango de operación de 0-300 grados y necesitan una alimentación de 10V a 12V.

La configuración inicial de estos servomotores se logró utilizando un dispositivo convertidor de USB a bus Dynamixel (llamado "USB2Dynamixel) y el programa Roboplus. En este programa se configuró la velocidad de comunicación, configuración serial y el identificador de cada servo.

Figura 4. Servomotores RX-10 de la marca Dynamixel

Se diseñó una pieza mecánica para ser añadida al final del brazo robótico y actuar como sujetador de herramienta. El diseño de esta herramienta se hizo mediante software CAD y se imprimió en 3D en plástico PLA. Como se observa en la Figura 5, la pieza consta de 2 partes que se unieron luego entre sí. El objetivo de este sujetador es añadir una herramienta con punta para facilitar la demostración de las coordenadas en el espacio alcanzadas en el proyecto mediante las rutinas que se explicarán en las siguientes secciones.

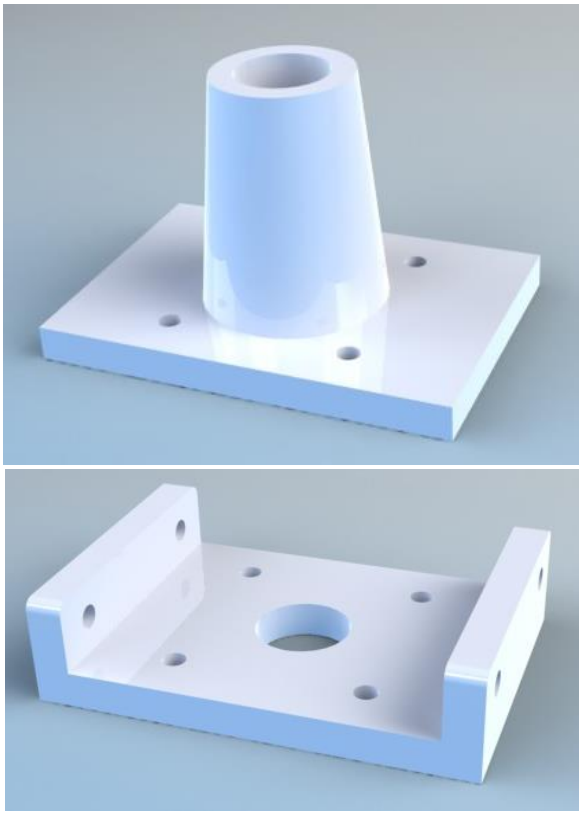

Figura 5. Pieza mecánica diseñada para actuar como sujetador de herramienta.

2.2. Control eléctrico

Para desarrollar el control eléctrico del brazo robótico se utilizó la placa BeagleBone Black (ver Figura 6). Como se mencionó en el apartado anterior, la comunicación con los servomotores RX-10 se realiza mediante comunicación serial; por lo que se utilizó una de las "Universal Asynchronous Receiver/Transmitter" o UART que posee la placa, específicamente la UART 2 ya que como se verá más adelante esta está conectada con el convertidor de protocolo de la placa RoboCape. La comunicación por UART se caracteriza porque tiene 3 pines básicos: TX (transmisión), RX (recepción) y GND (tierra), no posee señal de reloj (de ahí el término asincrónico). En el caso de la tarjeta BeagleBone Black los valores de voltaje a la salida de la UART son de 0V a 3.3V.

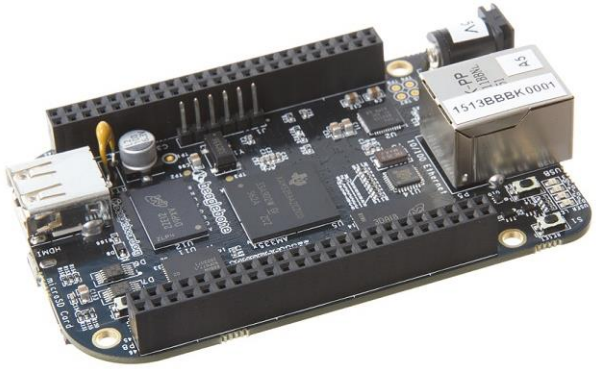

Figura 6. Placa BeagleBone Black utilizada en el proyecto.

Los servomotores RX-10 de la marca Dynamixel se controlan mediante comunicación serial con el protocolo RS-485. En el código implementado en el lenguaje C++ para el proyecto, se utilizó la librería BlackLib la cual permite utilizar la UART con tramas RS485. Sin embargo las conexiones eléctricas no son las mismas entre los servos y la BeagleBone Black. Los RX-10 se comunican mediante RS-485 half-duplex, en el cual utilizan 2 líneas diferenciales (D+ y D-) para enviar y recibir datos (en diferentes instantes) y con voltajes de -5V a 5V. En cambio, como se mencionó antes, la placa BeagleBone utiliza una línea para enviar datos (Tx) y otra para recibir datos (Rx); y estas tienen valores de 0V a 3.3V. En la siguiente imagen (Figura 7) se puede apreciar las diferencias para la misma trama de datos pero con ambos tipos de conexiones.

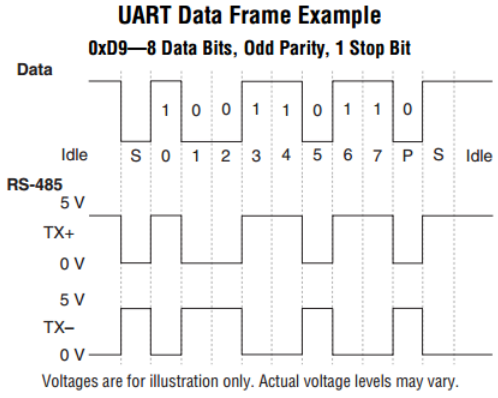

Figura 7. Diferencias entre datos UART TTL y RS-485.[1]

[1] Imagen tomada de la página http://www.piclist.com/images/UARTDataExample.png

Para solventar las mencionadas diferencias entre conexiones, se utilizó la placa RoboCape, la cual está especialmente diseñada para ser conectada a la BeagleBone Black y posee diversos periféricos que resultan de utilidad en la implementación y control de proyectos de Robótica. En este caso se utilizó el componente MAX485 que posee la placa, el cual se encarga de convertir las conexiones de UART TTL a RS-485 Diferencial. Se conectó las salidas de la RoboCape correspondientes a este periférico (conector J5) directamente al bus de datos que se conecta a los servomotores. Esta conexión se puede entender también mediante el diagrama que se muestra en la Figura 8.

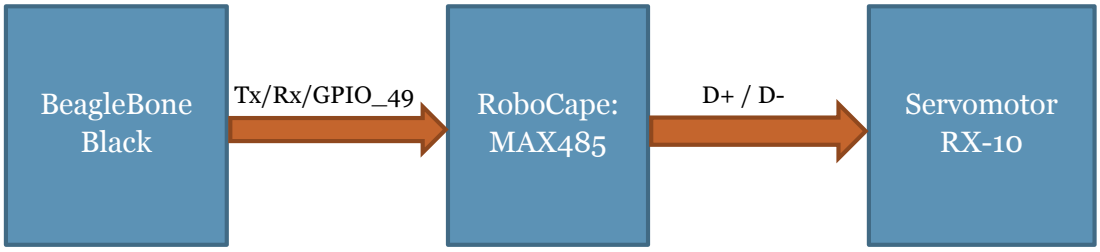

Figura 8. Diagrama de conexiones de la placa BeagleBone Black a los servomotores RX-10.

Un reto que fue necesario solventar durante la ejecución del proyecto y que es importante recalcar es que el dispositivo MAX485 necesita una línea que indique si se va a enviar de la UART (señal en Alto) o a recibir (señal en bajo). Esta señal se encuentra conectada al pin GPIO_49 de la placa BeagleBone Black y es necesario configurarla en alto o bajo según se necesite durante la ejecución del programa.

El envío de comunicaciones exitosas mediante esta conexión se probó utilizando el dispositivo USB2Dynamixel y un programa de híper terminal. En este ejercicio, se conectó el dispositivo al bus de los servomotores, luego se enviaron valores por la UART y se observó la aparición en la híper terminal de dichos valores. Así mismos se enviaron datos en el sentido contrario, de la híper terminal hacia la UART.

2.3. Instrucciones al servomotor

Una vez que se solventaron las conexiones eléctricas entre la placa controladora y los servomotores, se desarrollaron en C++ las rutinas para poder enviar un dato según el formato definido por el fabricante. Cuando se desea indicarle al servo a que posición ir, lo que se hace es enviar una instrucción de escritura al registro de "Goal position" con el valor deseado.

El formato de instrucción que se envía al servo se puede apreciar en la Figura 9, donde cada recuadro corresponde a un dato de tamaño 1 byte. Los 2 primeros bytes corresponden al valor 0xFF, el tercer byte es el número que identifica a cada servomotor. Seguidamente se envía la longitud de del paquete, la cual se calcula como el número de parámetros más 2. El siguiente byte es la instrucción a enviar, que en el caso de escritura es 0x03. Luego se envía la dirección del registro donde se desea escribir, en el caso de la posición deseada es el registro 30 o 0x1E. Los siguientes 2 bytes corresponden al valor de posición deseado, para esto se debe dividir el valor en 2 bytes, enviar primero el byte bajo del valor y luego el byte alto del valor. El último valor a enviar corresponde a un valor para revisar si la transmisión sucedió sin errores; este valor se calcula como el inverso del resultado de la suma de todos los bytes del mensaje (excluyendo los 0xFF).

0xFF	0xFF	ID	Longitud	Instrucción 0x03	Dirección escritura 0x1E	Valor (byte bajo)	Valor (byte alto)	Check Sum

Figura 9. Formato paquete de escritura de posición en el servomotor.

En el proyecto se implementó la creación de este mensaje mediante una función llamada AXServoMove. Esta rutina recibe el número de identificación del servo y la posición deseada. Luego la posición se descompone en 2 bytes (alto y bajo) y se arma el mensaje según el formato anteriormente indicado. Esta rutina retorna un valor tipo String que luego debe ser escrito a la UART en la rutina principal de la aplicación.

Función	AXServoMove
Descripción	Crea un mensaje para mover un servo RX-10
Argumento de entrada	Entero (int) con el identificar del servo
Argumento de entrada	Entero (int) con la posición deseada (de 0 a 300 grados)
Retorna	Cadena de caracteres (String) correspondiente al mensaje

Tabla 1. Descripción de la función AXServoMove

Para el cálculo del CheckSum se implementó una función genérica que puede servir para cualquier tipo de comunicación con servos Dynamixel. Esta función recibe el ID, la Longitud, la Instrucción y un vector con los parámetros (en el caso de escritura: la dirección y el valor); realiza la suma de todos estos valores y luego realiza la inversión del valor. En caso de que el resultado fuera un valor mayor a 1 byte, la función trunca este valor a un solo byte. Esta función retorna el valor correspondiente al CheckSum como un entero. Los parámetros se reciben como un vector

de caracteres debido a que la cantidad de parámetros es variable según la instrucción que se planea realizar y la zona de memoria sobre la que se desea escribir o leer.

Función	AXGetCheckSum
Descripción	Calcula el Checksum de un mensaje para un servo RX-10
Argumento de entrada	Entero (int) con el identificar del servo
Argumento de entrada	Entero (int) con la longitud del mensaje
Argumento de entrada	Entero (int) con la instrucción
Argumento de entrada	Vector de caracteres (char) con los parámetros
Retorna	Entero (int) con el valor del Checksum

Tabla 2. Descripción de la función AXGetCheckSum

Se debe señalar un aspecto que provocó múltiples confusiones durante el desarrollo del proyecto y que debió ser solucionado apropiadamente. Al crear un valor tipo String con valores enteros y hexadecimales, operadores como "<<" pueden ocasionar que el valor se convierta a ASCII primero y luego se añada al String. Debido a que los datos son de tamaño 1 byte lo más conveniente y lo que así se hizo en el proyecto fue convertirlos a un valor tipo Char primero y luego añadirlos al String.

2.4. Cinemática Inversa

Por Cinemática Inversa de un brazo robótico se entiende el cálculo de los valores de los ángulos de las articulaciones que logran que la punta del brazo alcance un punto en el espacio determinado. Para este cometido, el proyecto se basó en el trabajo de Núñez et al en "Explicit analytic solution for inverse kinematics of Bioloid humanoid robot".

En esa investigación se propone una solución analítica explícita para el problema de resolver la cinemática inversa, donde dada la facilidad que aporta que el brazo sólo posea 3 grados de libertad, se permite el desarrollo de ecuaciones que describen en cualquier instante las posiciones de los ángulos. En el caso de brazos robóticos más complejos, como es el caso de los brazos con 6 grados de libertad, el cálculo de la cinemática inversa resulta computacionalmente más intenso, teniendo que recurrir a herramientas como aproximaciones sucesivas para resolver algebra matricial.

Para este proyecto se utilizó la misma convención de ejes que en la mencionada investigación, ya que uno de los objetivos es poder extender luego el presente trabajo a un robot humanoide. Esta convención toma como el eje *x* con una orientación hacia afuera del pecho (línea azul en la Figura 10, segunda imagen), el eje *y* hacia la izquierda del robot humanoide (línea roja) y el eje *z* hacia la cabeza (línea verde). En el caso del brazo robótico, este representa el brazo derecho del robot humanoide, por tanto (como se observa en la Figura 10, primera imagen) el eje *x* tendría una orientación hacia la derecha del brazo, el eje *z* hacia el frente y el eje *y negativo* (*-y*) hacia arriba del brazo.

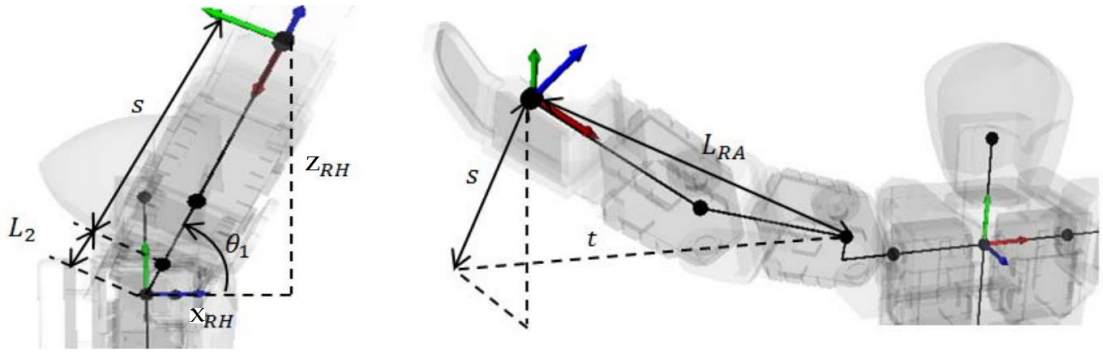

Figura 10. Ejes de referencia y nomenclaturas usadas para la cinemática inversa[2].

El primer ángulo que se calcula es el de la base del brazo robótico, que en la investigación se refiere como θ_1, y este viene dado por la proyección de la punta del brazo sobre el eje *x* y la proyección sobre el eje *z* (ver Figura 10, primera imagen). Se utilizó la función atan2 en el código ya que tiene la ventaja de que calcula el arco tangente y también tiene en consideración los signos de los argumentos para calcular la ubicación del ángulo resultante. La fórmula utilizada se muestra en la ecuación 1.

$$\theta_1 = \text{atan2}(z_{RH}, x_{RH}) \qquad \text{(Ecuación 1)}$$

[2] Imagen tomada de "Explicit analytic solution for inverse kinematics of Bioloid humanoid robot" de Nunez et al.

Para el cálculo del ángulo del codo del brazo robótico, se utilizó la proyección de la punta del brazo sobre el eje *y*, en la investigación se refiere a este valor como *t* (ecuación 2). También se utilizó la hipotenusa entre las proyecciones sobre *x* y sobre *y*, la cual es referida como s (ecuación 3). Debe notarse que dado que el eje de referencia se toma como en el centro del pecho del robot, en las ecuaciones de *t* y *s* se restan las longitudes para llegar al hombro del robot. En el caso del brazo robótico, estas longitudes corresponden a las distancias de la plataforma sobre la que está montada la base. Posteriormente con los valores de s y t se calculó el valor de la extensión del brazo, referida como L_{RA}, y dada por la ecuación 4.

$$t = -y_{RH} - L_0 - L_1 \quad \text{(Ecuación 2)}$$

$$s = \sqrt{x_{RH}^2 + z_{RH}^2} - L_2 \quad \text{(Ecuación 3)}$$

$$L_{RA} = \sqrt{s^2 + t^2} \quad \text{(Ecuación 4)}$$

Como ya se poseen todas las longitudes del triángulo formado por el antebrazo, brazo y extensión del brazo, por ley de cosenos se puede obtener el valor del coseno del ángulo del codo, el cual es referido como $cos(\gamma_{RA})$ y es referido también como C_5. Este valor está dado por la ecuación 5. El cálculo del valor del ángulo en el codo está dado por la ecuación 6.

$$\cos(\gamma_{RA}) = \frac{L_{RA}^2 - L_3^2 - L_4^2}{2L_3L_4} \quad \text{(Ecuación 5)}$$

$$\theta_5 = -\text{atan2}\left(\sqrt{1 - {C_5}^2}, C_5\right) \quad \text{(Ecuación 6)}$$

Para obtener el ángulo del hombro, se utilizan 2 ángulos auxiliares; el primer es referido como γ_1 y está formado por *s*, *t* y la extensión del brazo L_{RA} (ver ecuación 7), el segundo ángulo es referido como γ_1 y está formado por L_{RA} y la extensión del brazo L_3 (ver ecuación 8). En la Tabla 3 se pueden observar los valores de longitud de las partes del brazo que se utilizaron y cuál es su denominación en las ecuaciones. Finalmente el valor del ángulo del hombro y referido como θ_3, está dado por la diferencia de los ángulos anteriormente encontrados (ecuación 9).

$$\gamma_1 = \text{atan2}(s, t) \quad \text{(Ecuación 7)}$$

$$\gamma_2 = \text{atan2}(L_4\sin(\theta_5), L_3 + L_4\cos(\theta_5)) \quad \text{(Ecuación 8)}$$

$$\theta_3 = -(\gamma_1 - \gamma_2) \quad \text{(Ecuación 9)}$$

Denominación	Medida (mm)	Descripción
L_0	0.073	Distancia del origen al eje del servo de la base
L_1	0.026	Distancia del eje del servo de la base al eje del servo del hombro
L_2	0	Distancia del servo del hombro al eje y (En el robot humanoide este servo y el eje no están alineados, es decir L2>0)
L_3	0.067	Longitud del brazo: Distancia del eje del servo del hombro al eje del servo del codo
L_4	$0.039+l_{tool}$	Longitud del antebrazo: Distancia del eje del servo del codo al borde del servo del codo. Se le suma la longitud de la herramienta
L_{tool}	0.081	Longitud de la herramienta utilizada

Tabla 3. Longitudes del brazo utilizadas

Para probar las ecuaciones cinemáticas mencionadas en la investigación se utilizó el programa Matlab con la Robotics ToolBox (hecha por Peter Corke y explicada en el libro Robotics, Vision and Control). Con esto se creó una cadena cinemática de eslabones a los cuales se les pudo aplicar las ecuaciones anteriormente descritas (ver Figura 11), hay que notar que los ejes en Matlab se muestran en un colocación diferente a la usada en el proyecto. Mediante esta simulación se pudo comprobar una discrepancia en los planteamientos de la investigación, ya que los ángulos que aparecían como negativos en las imágenes, se tomaban como positivos en las ecuaciones. Además esta simulación permitió tener certeza de cómo ajustar los ángulos obtenidos a la nomenclatura de ángulos utilizada en el proyecto.

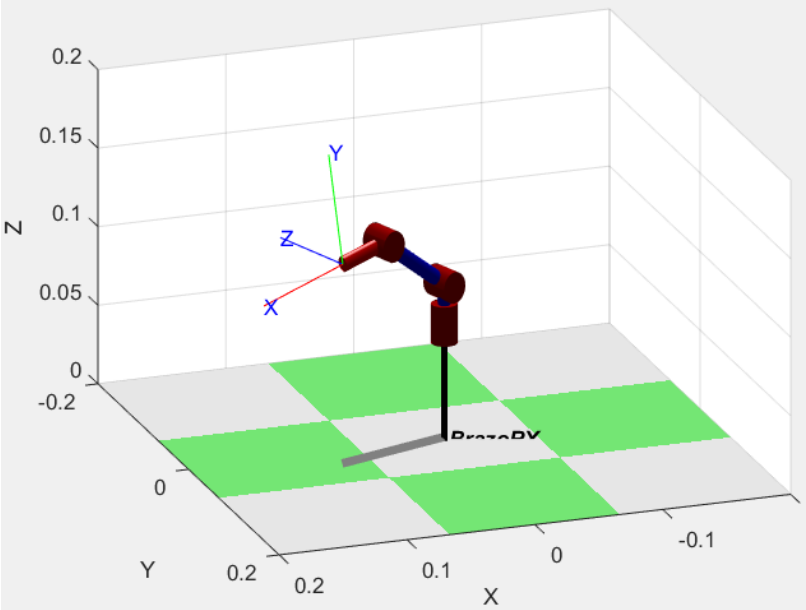

Figura 11. Simulación en Matlab del brazo robótico.

En el proyecto, la ecuaciones de cinemática inversa se implementaron mediante una función llamada IK3ServoArm. Esta rutina recibe las coordenadas de un punto en el espacio y las dimensiones del brazo. Posteriormente utiliza todas las ecuaciones mencionadas y retorna los ángulos de cada articulación que alcanzan el punto en el espacio recibido. Los ángulos se retornan por referencia mediante un parámetro de salida en los argumentos.

Función	IK3ServoArm
Descripción	Calcula la cinemática inversa de un brazo de 3 grados de libertad
Argumento de entrada	Real (float) con la coordenada x del punto en el espacio
Argumento de entrada	Real (float) con la coordenada y del punto en el espacio
Argumento de entrada	Real (float) con la coordenada z del punto en el espacio
Argumento de entrada	Vector (float) con las longitudes de las partes del brazo
Argumento de salida	Vector (int) con los ángulos calculados

Tabla 4. Descripción de la función IK3ServoArm.

Debido a que los servomotores Dynamixel tienen una nomenclatura de ángulos diferente a la usada en la investigación, fue necesario crear una función que convirtiera los valores de ángulos obtenidos en la cinemática inversa, los cuales corresponden al número de grados en que se desvía el motor de su eje *y* de coordenadas. El formato Dynamixel tiene como origen 0 grados, tiene 150 grados alineado con el eje *y*, y tiene como final 300 grados.

Además en esta función se implementó revisiones de posibles errores, como en el caso de que el codo y/o el hombro se les asignara un ángulo que ocasione un choque con el soporte metálico (ángulo < 60 grados o ángulo > 240 grados), en cuyo caso el ángulo se satura a su valor límite. Otro caso de error corresponde cuando el ángulo tiene un valor menor a cero o mayor a 300 grados, esto significa que el punto al que se desea llegar no es alcanzable y por ende se muestra un mensaje de error.

Esta función se llamó IKAdjust3RX y recibe un vector con los 3 ángulos a corregir; retorna por referencia los ángulos corregido en el mismo vector.

Función	IKAdjust3RX
Descripción	Transforma los ángulos obtenidos en la cinemática inversa a la nomenclatura Dynamixel. Corrige posibles valores máximos en los ángulos.
Argumento de entrada/salida	Vector (int) con los ángulos a ser corregidos. Retorna el mismo vector con los ángulos corregidos.

Tabla 5. Descripción de la función IKAdjust3RX.

2.5. Generación de Trayectorias

Una vez que se logró que el brazo robótico alcanzara un punto específico en el espacio, el siguiente paso consistió en lograr que el robot se desplazara entre dos puntos en el espacio describiendo una trayectoria deseada. Se implementaron 2 tipos de trayectorias en el proyecto: trayectoria libre y trayectoria lineal (o cartesiana).

En la trayectoria libre, la punta del brazo robótico se desplaza entre 2 puntos de la manera más rápida y fácil posible para sus articulaciones. Generalmente este tipo de trayectorias describe un movimiento circular o de arco (ver Figura 12), ya que al ser las articulaciones de tipo revolución, este es el movimiento más simple posible.

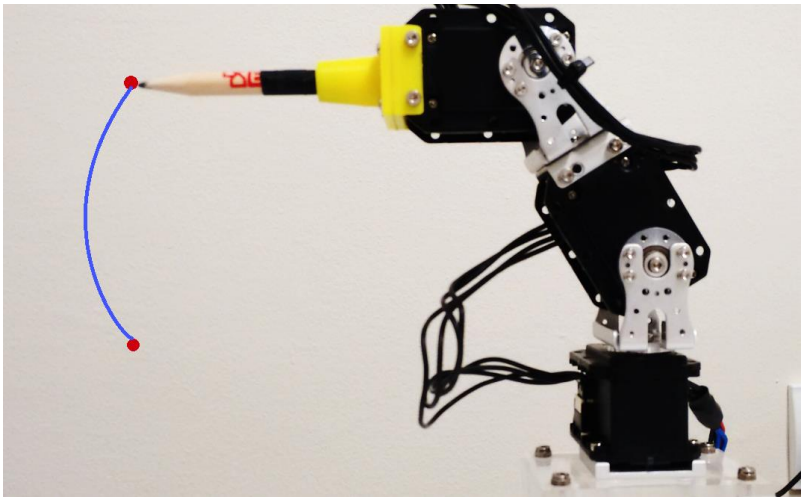

Figura 12. Ilustración de una trayectoria libre para el brazo robótico.

Para lograr describir este tipo de movimiento se implementó la función freeTraj, la cual recibe el punto en el espacio donde se va a iniciar el movimiento y el punto donde va a terminar; y también recibe las longitudes de las partes del brazo (necesarias para la cinemática inversa). Con estos datos, la función calcula la cinemática inversa para el punto de inicio y también para el punto final. Luego para cada articulación se toman los ángulos de inicio y de fin y se calculan una serie de ángulos intermedios entre estos 2 ángulos. La cantidad de ángulos intermedios está determinado por la variable global llamada "sample". Con esos ángulos obtenidos, la función crea una matriz con los valores para cada articulación en cada instante de tiempo, la cual es retornada por referencia.

Función	freeTraj
Descripción	Calcula una trayectoria libre entre 2 puntos
Argumento de entrada	Vector (float) con las coordenadas del punto de inicio
Argumento de entrada	Vector (float) con las coordenadas del punto de fin
Argumento de entrada	Vector (float) con las longitudes de las partes del brazo
Argumento de salida	Matriz (int) con los ángulos para cada instante de tiempo de la trayectoria

Tabla 6. Descripción de la función freeTraj.

En una trayectoria lineal o llamada también trayectoria cartesiana, la punta del brazo robótico describe una trayectoria que corresponde a una línea recta en el espacio cartesiano (ver Figura 13). Este tipo de movimiento es el que puede resultar más intuitivo al pensar en trayectorias, pero a la vez es mucho más complicado para el robot ya que debe variar las posiciones de sus articulaciones durante el movimiento para lograr que la punta del brazo permanezca en la trayectoria de la línea recta.

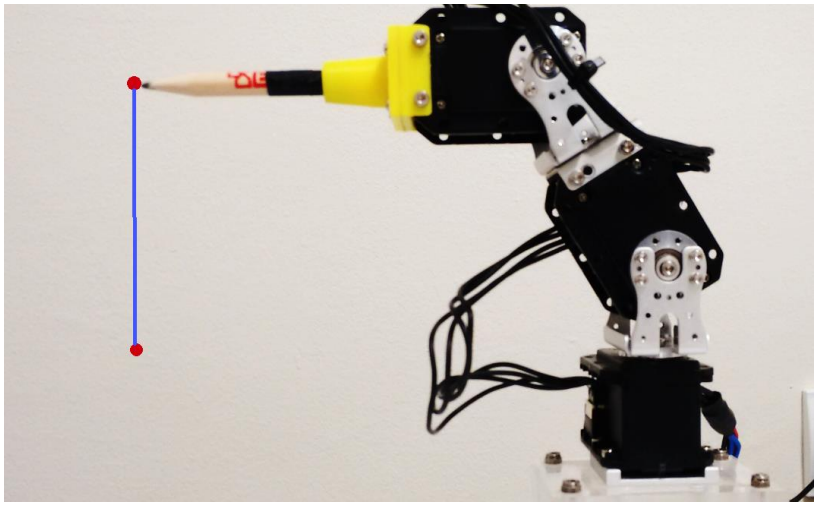

Figura 13. Ilustración de una trayectoria lineal o cartesiana para el brazo robótico

En el proyecto, para crear una trayectoria lineal o cartesiana se implementó la función linearTraj, la cual recibe el punto de inicio de la trayectoria lineal en el espacio y el punto de fin de la trayectoria; y también recibe las longitudes de las partes del brazo. Con estos valores de entrada, la función crea una línea entre el punto de inicio y el punto final: luego divide esta línea en una serie de puntos intermedios. Así como en el caso de la función freeTraj, en la función linearTraj la cantidad de puntos intermedios o divisiones de la línea está establecido por la variable global "sample". Para cada uno de los puntos intermedios obtenidos la función calcula la cinemática inversa del brazo robótico, y almacena dichos valores en una matriz. Esta matriz se retorna por referencia y contiene los valores de ángulos para cada articulación en cada instante de tiempo.

Función	linearTraj
Descripción	Calcula una trayectoria linear o cartesiana entre 2 puntos
Argumento de entrada	Vector (float) con las coordenadas del punto de inicio
Argumento de entrada	Vector (float) con las coordenadas del punto de fin
Argumento de entrada	Vector (float) con las longitudes de las partes del brazo
Argumento de salida	Matriz (int) con los ángulos para cada instante de tiempo de la trayectoria

Tabla 7. Descripción de la función linearTraj.

2.6. Selección de parámetros para ejecución

En el proyecto se crearon diversas funciones para mover el brazo robótico: según los ángulos, según un punto en el espacio, o según una determinada trayectoria. En esta última etapa se buscó el crear la manera de que estas funciones fueran accesibles por el usuario para ser usadas de una manera fácil sin tener que acceder necesariamente al código fuente del programa.

Mediante los parámetros con que se llama a la aplicación se creó una manera de poder acceder a dichas funciones creadas y por ende lograr este tipo de interacción deseado. En la Tabla 8 se especifican los parámetros necesarios para poder acceder a las funciones mediante llamadas de consola a la aplicación. El último caso (-d) corresponde a una secuencia de demostración que se programó en el robot, en la cual la punta del robot escribe la palabra "HOLA" en un plano vertical para el brazo (plano en el eje –y).

Parámetro Inicial	Otros parámetros	Descripción
-r	*ninguno*	Ir a la posición de descanso
-a	<ang1> <ang2> <ang3>	Aplicar ángulos a los servos
-p	<posx> <posy> <posz>	Ir a la posición especificada
-f	<px1> <py1> <pz1> <px2> <py2> <pz3>	Trayectoria libre de p1 a p2
-l	<px1> <py1> <pz1> <px2> <py2> <pz3>	Trayectoria libre de p1 a p2
-d	*ninguno*	Ejecución de secuencia de demostración

Tabla 8. Descripción de los parámetros para ejecutar funciones por consola.

Este tipo de ejecución por consola permite que un usuario con sólo estar conectado a la tarjeta BeagleBone Black y tener los permisos necesarios, pueda controlar e interactuar con el robot de una manera sencilla como se observa en los ejemplos de la Figura 14. Incluso con esta metodología, se pueden programar secuencias de movimientos más complejas mediante Scripts de Shell.

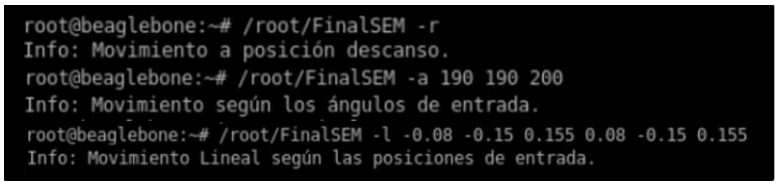

Figura 14. Ejemplos de llamadas por consola a funciones de la aplicación.

3. Conclusiones

En este proyecto se puso en práctica los conocimientos adquiridos a través de las sesiones de teoría y práctica de la asignatura Sistemas Empotrados, como lo son la configuración de un ambiente de desarrollo y compilación cruzada, la conexión a través de SSH con un dispositivo empotrado, el manejo de los pines GPIO de la tarjeta BeagleBone Black, la comunicación por medio de UART y el uso de la tarjeta RoboCape. También se aplicaron conocimientos adquiridos en otras asignaturas del master respecto a programación y control de robots.

En primer lugar se armó la estructura mecánica de un brazo robótico de 3 grados de libertad (DOF) utilizando una plataforma, servomotores Dynamixel R-10, conectores de aluminio y un sujetador de herramienta. Esta última pieza se diseñó mediante herramientas CAD y fue impresa en 3D para poder sujetar herramientas cilíndricas.

Luego se trabajó en la conexión de la placa BeagleBone Black con los servomotores, ya que a pesar de que ambos pueden trasmitir tramas de protocolo RS485, la conexión eléctrica es diferente. Por eso se utilizó la placa RoboCape, específicamente el periférico MAX485 que provee la facilidad de convertir dichas señales eléctricas entre ambos dispositivos.

Para mover el servo, se implementó una función que crea un mensaje para ser transmitido al servomotor y que escribe en el registro “Goal Position” el valor de posición deseado. Esta función a su vez utiliza la función para calcular el valor de Checksum en el mensaje, y que también fue desarrollada para el proyecto.

El proyecto se basó en la investigación “Explicit analytic solution for inverse kinematics of Bioloid humanoid robot” de Núñez et al para la implementación de las ecuaciones de cinemática inversa del brazo robótico. Dichas ecuaciones fueron simuladas y probadas primeramente en Matlab utilizando el Robotics ToolBox. Luego se implementó una función que recibe un punto en el espacio y utilizando las ecuaciones calcula el valor de los ángulos de cada articulación que logran que la punta del brazo alcance dicho punto. Fue necesario crear otra función para ajustar los valores de los ángulos entre la convención utilizada en la investigación y la nomenclatura de ángulos utilizada por los servomotores Dynamixel.

Para mover el brazo entre 2 puntos en el espacio mediante una trayectoria específica se crearon 2 funciones: una para crear una trayectoria libre y otra para crear una trayectoria lineal o cartesiana.

Finalmente se creó una serie de parámetros con que la aplicación al ser llamada por consola puede ejecutar cada una de las funciones creadas. Esta es una manera fácil y útil de controlar el brazo robot con sólo conectarse a la placa BeagleBone Black.

Como continuación para los temas tratados en este proyecto, se tiene el desarrollar el mismo proceso de control, mensajes, cinemática inversa, trayectorias y parámetros de ejecución para las demás partes de un robot humanoide. Estos temas serán desarrollados en el trabajo de Fin de Máster y generarán un compendio de funciones para la locomoción de un robot Bioloid.

4. Anexos

4.1. Código del proyecto (FinalSEM.cpp)

```
//*****************************************************************************
//*** Control de Brazo Robótico de 3 Grados de Libertad                     ***
//*** Utiliza servos RX-10. Control mediante BleagleBone Black y Robocape   ***
//*** Funciones para mover el brazo según ángulo y posición. Así como para  ***
//*** desplazarse en trayectoria lineal o libre                             ***
//*** Desarrollado por Juan Carlos Brenes. Mayo 2016. UPV, Valencia.        ***
//*****************************************************************************

#include <errno.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <iostream>
#include <math.h>
#include <iomanip>
#include "BlackLib.h"
#include "GPIO.h"
#include <bitset>
//#include <stdexcept>

using namespace std;
using namespace BlackLib;
using namespace exploringBB;

//ID de los servos
const unsigned int idServoHombro1= 60;
const unsigned int idServoHombro2= 61;
const unsigned int idServoCodo= 62;
//Dimensiones de la herramienta y el brazo
const float longTool= 0.081;
float longitudesBrazo[5] = {0.073, 0.026, 0, 0.067, (0.039+longTool)};
unsigned int servoID[]= {idServoHombro1, idServoHombro2, idServoCodo};

const int sample=30;  //cantidad de divisiones para la trayectoria
const int twait=30; //tiempo de espera en mseg

int angDescanso[3]={150,110,240}; //Ángulos para la posición de descanso del brazo

//Puntos para implementar la trayectoria de la letra H
float posAtaqueH1[3]={-0.055, -0.15, 0.125};
float posAtaqueH2[3]={-0.035, -0.15, 0.125};
float posAtaqueH3[3]={-0.055, -0.135, 0.125};
float posH1i[3]={-0.055, -0.15, 0.155};
float posH1f[3]={-0.055, -0.12, 0.155};
float posH2i[3]={-0.035, -0.15, 0.155};
float posH2f[3]={-0.035, -0.12, 0.155};
float posH3i[3]={-0.055, -0.135, 0.155};
float posH3f[3]={-0.035, -0.135, 0.155};

//Puntos para implementar la trayectoria de una letra O cuadrada.
float posAtaqueO[3]={-0.025, -0.15, 0.125};
float posO1i[3]={-0.025, -0.15, 0.155};
float posO1f[3]={-0.025, -0.12, 0.155};
float posO2f[3]={-0.005, -0.12, 0.155};
float posO3f[3]={-0.005, -0.15, 0.155};

//Puntos para implementar la trayectoria de una letra L.
float posAtaqueL[3]={0.005, -0.15, 0.125};
float posL1i[3]={0.005, -0.15, 0.155};
float posL1f[3]={0.005, -0.12, 0.155};
float posL2f[3]={0.025, -0.12, 0.155};
```

```
//Puntos para implementar la trayectoria de la letra A cuadrada
float posAtaqueA2[3]={0.055, -0.15, 0.125};
float posAtaqueA1[3]={0.035, -0.15, 0.125};
float posAtaqueA3[3]={0.055, -0.135, 0.125};
float posA2i[3]={0.055, -0.15, 0.155};
float posA2f[3]={0.055, -0.12, 0.155};
float posA1i[3]={0.035, -0.15, 0.155};
float posA1f[3]={0.035, -0.12, 0.155};
float posA3f[3]={0.055, -0.135, 0.155};
float posA3i[3]={0.035, -0.135, 0.155};

//Función que calcula el checksum del mensaje a ser enviado al servo.
//Recibe el id, longitud, instrucción y vector con los parámetros
//Regresa un entero positivo con el valor del chucksum
unsigned int AXGetCheckSum (int id, int len, int inst, char param[]){

    unsigned int chksm= id + len + inst;

    for (unsigned int i=0; i < (sizeof(param)-1); i++) {
        chksm=chksm + (int)param[i];
    }
    chksm=~(chksm % 256); //se hace operacion NOT, y como un int es de 4 bytes, se hace
modulo por 256 para obtener solo el byte menos significativo
    return chksm;
}

//Muestra en la consola un string en formato binario
void showMsgBinary (string msg){

    std::cout << "Info: Mensaje en binario: ";
    for (unsigned int i = 0; i < msg.size(); i += 2) {
        std::cout << std::bitset<8>(msg[i]) << " ";
        std::cout << std::bitset<8>(msg[i + 1]) << " ";
    }
    std::cout << endl;

}

//Funcion para mover los servos. Recibe el id del servo y la posición deseada
//Retorna un string con el mensaje a ser enviado por la UART
string AXServoMove (unsigned int  id, unsigned int pos){

    unsigned int instruction= 0x03; //Instruccion de escritura en el servo
    unsigned int address= 0x1E; //Direccion de Goal Position en el servo

    if ((pos<0) or (pos>300)){ //la posicion maxima es 300 y la minima 0
        std::cout << "Error: Posición deseada fuera de rango. Rango permitido 0-300
grados."<<endl;
    }

    unsigned int value = round(pos * 3.41); //Convierte la pos a un valor de 0-1023
    unsigned int valueHigh= round(value / 256); //obtiene el byte superior del valor de
posicion
    unsigned int valueLow= value % 256;         //obtiene el byte inferior del valor de
posicion
    char parameters[]={char(address), char(valueLow), char(valueHigh)}; //almacena la
dirección y el valor de posición en un vector. Esto debido a que los parámetros pueden
ser variables
    unsigned int length= sizeof(parameters)+2;  //calcula la longitud del paquete en
base a la cantidad de parámetros
    unsigned int checksum = AXGetCheckSum(id, length, instruction, parameters);

    std::ostringstream buff; //buffer de salida
    //agrega al buffer el paquete de instruccion para el servo
    buff <<
char(0xFF)<<char(0xFF)<<char(id)<<char(length)<<char(instruction)<<parameters[0]<<parame
ters[1]<<parameters[2]<<char(checksum);

    //***DEBUG: Descomentar para ver la trama binaria enviada
    //showMsgBinary(buff.str());

    return buff.str();
}

//Función que calcula la cinemática inversa del brazo. Recibe los puntos x, y, z del
espacio.
```

```
//Modifica por referencia un vector con el valor de los ángulos en grados
void IK3ServoArm (float xh, float yh, float zh, float armLen[5], int ang[3]){

    std::cout << "Info: X= "<<xh << "  Y= "<<yh << "  Z= "<<zh <<endl;

    //Ecuación para encontrar el ángulo del servo del hombro horizontal
    int servoBase = round( atan2(zh,xh)* 180 / M_PI );
    ang[0]=servoBase;

    //Ecuaciones para encontrar el ángulo del servo del codo
    float s = sqrt(pow(xh,2) + pow(zh,2))-armLen[2];
    float t = -yh - armLen[0] - armLen[1];
    float Lra= sqrt(pow(s,2) + pow(t,2));
    float C5=  (pow(Lra,2) - pow(armLen[3],2) - pow(armLen[4],2)) / (2 * armLen[3] *
armLen[4]);
    float angleElbowRad = -atan2( sqrt(1-pow(C5,2)), C5 );
    int servoElbow= round(angleElbowRad * 180 / M_PI );
    ang[2]=servoElbow;

    //Ecuaciones para encontrar el ángulo del servo del hombro vertical
    float gamma1= atan2(s,t);
    float gamma2= atan2( (armLen[4]*sin(-angleElbowRad)), (armLen[3]+armLen[4]*cos(-
angleElbowRad)));
    int servoShoulder = round( -(gamma1-gamma2) * 180 / M_PI );
    ang[1]=servoShoulder;

}

//Función que recibe 3 ángulos calculados de la cinemática inversa y los ajusta a las
especificaciones del robot Bioloid
//y los servos RX10. Modifica por referencia un vector con el valor de los ángulos en
grados
void IKAdjust3RX (int ang[3]){

    //Cuando los 2 ángulos son cero, en casi todos los casos corresponde a un punto no
alcanzable
    if ((ang[1]==0) || (ang[2]==0)){
            std::cout << "Error: La posición en el espacio no es alcanzable."<<endl;
    }

    //Ajusta el ángulo de la base en base.
    int servoBaseRX10= 152-(90-ang[0]); //150: ajuste del RX, 90-: ajuste con el eje de
coordenadas
    if ((servoBaseRX10<0) || (servoBaseRX10>300)){
        std::cout << "Error: La posición deseada no es alcanzable. Error de valor en el
servo de la base."<<endl;
        ang[0]=150;  //Pone la posición por defecto del servo
    }else{
        ang[0]=servoBaseRX10;
    }

    ang[0]=servoBaseRX10;

    //Ajusta el ángulo del hombro
    int servoShoulderRX10= 150-(ang[1]); //150-: ajuste del RX
    //Revisa los valores máximos y minimos del servo
    if ((servoShoulderRX10<0) || (servoShoulderRX10>300)){
        std::cout << "Error: La posición deseada no es alcanzable. Error de valor en el
servo del hombro."<<endl;
        ang[1]=150;  //Pone la posición por defecto del servo
    }else if (servoShoulderRX10<60){
            std::cout << "Error: La posición deseada provoca colosión en el servo del
hombro. Servo saturado a su valor mínimo."<<endl;
        ang[1]=60;
    }else if (servoShoulderRX10>240){
        std::cout << "Error: La posición deseada provoca colosión en el servo del
hombro. Servo saturado a su valor máximo."<<endl;
        ang[1]=240;
    }else{
        ang[1]=servoShoulderRX10;
    }

    //Ajusta el ángulo del codo
    int servoElbowRX10= 150-(ang[2]); //150-: ajuste del RX
    //Revisa los valores máximos y minimos del servo
    if ((servoElbowRX10<0) || (servoElbowRX10>300)){
```

```
        std::cout << "Error: La posición deseada no es alcanzable. Error de valor en el
servo del codo."<<endl;
        ang[2]=150; //Pone la posición por defecto del servo
    }else if (servoElbowRX10<60){
            std::cout << "Error: La posición deseada provoca colisión en el servo del
codo. Servo saturado a su valor mínimo."<<endl;
        ang[2]=60;
    }else if (servoElbowRX10>240){
        std::cout << "Error: La posición deseada provoca colosión en el servo del codo.
Servo saturado a su valor máximo."<<endl;
        ang[2]=240;
    }else{
        ang[2]=servoElbowRX10;
    }

    std::cout << "Info: Ang Corregidos. Base: "<<ang[0]<< "  Hombro: "<<ang[1]<<"  Codo:
"<<ang[2]<<endl;
}

//Función que calcula una trayectoria lineal para la punta del brazo. Recibe el punto de
inicio y el punto final, así como las dimensiones del brazo.
//Recibe por referencia una matriz donde devuelve los ángulos para cada instante de
tiempo.
void linearTraj (float pos1[3], float pos2[3], float armLen[5], int trajAng[3][sample]){

    //Se ejecuta el proceso para cada instante de tiempo. La cantidad de instantes está
dada por la variable Sample
    for (int i=1;i<=sample;i++){
        //Se divide el trayecto entre la cantidad de instantes de tiempo. Obteniendo una
serie de puntos intermedios
        //Se recorren uno por uno estos puntos
        float pos_xi= ( (pos2[0]-pos1[0])/sample*i ) + pos1[0] ;
        float pos_yi= ( (pos2[1]-pos1[1])/sample*i ) + pos1[1] ;
        float pos_zi= ( (pos2[2]-pos1[2])/sample*i ) + pos1[2] ;
        //En cada punto del trayecto se calcula la Cinemática Inversa
        int angles[3]={0,0,0};
        IK3ServoArm (pos_xi, pos_yi, pos_zi, armLen, angles); //Calcula la Cinemática
Inversa
        IKAdjust3RX (angles);       //Corrige los ángulos para ajustarlos a los servos
RX
        //Se almacenan los ángulos para cada punto del trayecto lineal
        trajAng[0][i]= angles[0];
        trajAng[1][i]= angles[1];
        trajAng[2][i]= angles[2];
    }
}

//Función que calcula una trayectoria libre entre 2 puntos. Recibe el punto de inicio y
el punto final, así como las dimensiones del brazo.
//Recibe por referencia una matriz donde devuelve los ángulos para cada instante de
tiempo.
void freeTraj (float pos1[3], float pos2[3], float armLen[5], int trajAng[3][sample]){

    //Se calcula la Cinemática Inversa para el punto inicial
    int angles1[3]={0,0,0};
    IK3ServoArm (pos1[0], pos1[1], pos1[2], armLen, angles1); //Calcula la Cinemática
Inversa
    IKAdjust3RX (angles1);      //Corrige los ángulos para ajustarlos a los servos RX
    //Se calcula la Cinemática Inversa para el punto final
    int angles2[3]={0,0,0};
    IK3ServoArm (pos2[0], pos2[1], pos2[2], armLen, angles2); //Calcula la Cinemática
Inversa
    IKAdjust3RX (angles2);          //Corrige los ángulos para ajustarlos a los servos
RX

    //Se ejecuta el proceso para cada instante de tiempo. La cantidad de instantes está
dada por la variable Sample
    for (int i=1;i<=sample;i++){
        //Se divide el ángulo inicial y el final entre la cantidas de instántes de
tiempo. Obteniendo una serie de ángulos intermedios
        int angleBase_i= round( (float)(angles2[0]-angles1[0])/sample*i + angles1[0] ) ;
        int angleShoulder_i= round( (float)(angles2[1]-angles1[1])/sample*i + angles1[1]
);
        int angleElbow_i= round( (float)(angles2[2]-angles1[2])/sample*i + angles1[2] );
        //Se almacenan los ángulos para cada instante de tiempo
        trajAng[0][i]= angleBase_i;
        trajAng[1][i]= angleShoulder_i;
```

```
        trajAng[2][i]= angleElbow_i;
    }
}


int main(int argc, char *argv[]) {

    //Apertura e inicializacion de la UART
    BlackLib::BlackUART uartServos(
BlackLib::UART2,BlackLib::Baud38400,BlackLib::ParityNo,BlackLib::StopOne,BlackLib::Char8
);
    bool isOpened = uartServos.open( BlackLib::ReadWrite | BlackLib::NonBlock );
        if( !isOpened )     {
            std::cout << "UART DEVICE CAN\'T OPEN.;" << std::endl;
            exit(1);        }

        std::cout << std::endl;
        std::cout << "Device Path     : " << uartServos.getPortName() << std::endl;
        std::cout << "Read Buf. Size  : " << uartServos.getReadBufferSize() <<
std::endl;
        std::cout << "BaudRate In/Out : " << uartServos.getBaudRate( BlackLib::input) <<
"/"
                                          << uartServos.getBaudRate( BlackLib::output)
<< std::endl;
        std::cout << "Character Size  : " << uartServos.getCharacterSize() << std::endl;
        std::cout << "Stop Bit Size   : " << uartServos.getStopBits() << std::endl;
        std::cout << "Parity          : " << uartServos.getParity() << std::endl <<
std::endl;

    //Establecer la dirección de comunicación para el convertidor MAX485 de la RoboCape
    GPIO UART2 IO(49); //Pin que determina la dirección de la UART2
    UART2 IO.setDirection(GPIO::OUTPUT); //Se configura como salida
    UART2_IO.setValue(GPIO::HIGH);          //Enviar
    //UART_IO_49.setValue(GPIO::LOW);        //Recibir


    //Revisa que se tengan al menos 2 argumentos de entrada
    if (argc < 2){
        std::cout << "Error: Argumentos insuficientes. Se necesitan ingresar al menos un
paŕametro en la instrucción." << std::endl;
        exit(1);
    }

    //*** Descanso ***
    //El brazo se mueve hacia la posición de descanso
    if (strcmp (argv[1], "-r")==0){

        std::cout << "Info: Movimiento a posición descanso." << std::endl;
        for (int i=0; i<3; i++){
            //Se llama la funcion de mover Servo, la cual regresa un string que es el
mensaje a enviar
            string sal= AXServoMove(servoID[i], angDescanso[i]);
            uartServos.write(sal);
        }
    }

    //*** Movimiento a ángulos ***
    //El brazo se mueve a la posición especificada por los 3 ángulos del argumento
    else if (strcmp (argv[1], "-a")==0){

        std::cout << "Info: Movimiento según los ángulos de entrada." << std::endl;
        //Revisa que se tengan al menos 3 argumentos de entrada
        if (argc < 5){
            std::cout << "Error: Argumentos insuficientes. Se necesitan 3 valores de
ángulos." << std::endl;
            exit(1);
        }
        //Advierte si hay más argumentos de la cuenta
        if (argc > 5){
            std::cout << "Advertencia: Muchos argumentos. Se necesitan 3 valores de
ángulos. Se utilizarán sólo los 3 primeros argumentos." << std::endl;
        }
        //Revisa que los 3 argumentos sean números
        int ang[3];
        for (int i=2; i<5; i++) {
            if ( ! (istringstream(argv[i]) >> ang[i-2]) ) {
```

```
                std::cout << "Error: Argumento inválido. Debe ingresar un número
entero."<<endl;
                exit(1);
            }
        }
        for (int i=0; i<3; i++){
            //Se llama la funcion de mover Servo, la cual regresa un string que es el
mensaje a enviar
            string sal= AXServoMove(servoID[i], ang[i]);
            uartServos.write(sal);
        }
    }


    //*** Movimimiento a un punto ***
    //El brazo se mueve a la posicion especificada por las coordenadas x,y,z
    else if (strcmp (argv[1], "-p")==0){

        std::cout << "Info: Movimiento según la posición de entrada." << std::endl;
        //Revisa que se tengan al menos 3 argumentos de entrada
        if (argc < 5){
            std::cout << "Error: Argumentos insuficientes. Se necesitan 3 valores: x y
z" << std::endl;
            exit(1);
        }
        //Advierte si hay más argumentos de la cuenta
        if (argc > 5){
            std::cout << "Advertencia: Muchos argumentos. Se necesitan 3 valores de
posición. Se utilizarán sólo los 3 primeros argumentos." << std::endl;
        }
        //Revisa que los 3 argumentos sean números
        float pos[3];
        for (int i=2; i<5; i++) {
            if ( ! (istringstream(argv[i]) >> pos[i-2]) ) {
                std::cout << "Error: Argumento inválido. Debe ingresar un
número."<<endl;
                exit(1);
            }
        }
        //Se calcula la Cinemática Inversa para el punto ingresado
        int servoAngles[3]=  {0, 0, 0};
        IK3ServoArm (pos[0], pos[1], pos[2], longitudesBrazo, servoAngles); //Calcula la
Cinemática Inversa
        IKAdjust3RX (servoAngles);      //Corrige los ángulos para ajustarlos a los
servos RX
        for (int i=0; i<3; i++){
            //Se llama la funcion de mover Servo, la cual regresa un string que es el
mensaje a enviar
            string sal= AXServoMove(servoID[i], servoAngles[i]);
            uartServos.write(sal);
        }
    }

    //*** Movimiento Lineal ***
    //Se mueve el brazo en una trayectoria lineal
    else if (strcmp (argv[1], "-l")==0){

        std::cout << "Info: Movimiento Lineal según las posiciones de entrada." <<
std::endl;
        //Revisa que se tengan al menos 6 argumentos de entrada
        if (argc < 8){
            std::cout << "Error: Argumentos insuficientes. Se necesitan 6 valores:
coordenadas del punto inicial y coordenadas del punto final" << std::endl;
            exit(1);
        }
        //Advierte si hay más argumentos de la cuenta
        if (argc > 8){
            std::cout << "Advertencia: Muchos argumentos. Se necesitan 6 valores de
posición. Se utilizarán sólo los 6 primeros argumentos." << std::endl;
        }
        //Revisa que los 6 argumentos sean números
        float posIniLin[3];
        float posFinLin[3];
        for (int i=2; i<5; i++) {
            if ( ! (istringstream(argv[i]) >> posIniLin[i-2]) ) {
                std::cout << "Error: Argumento inválido. Debe ingresar un
número."<<endl;
```

```
                exit(1);
            }
        }
        for (int i=5; i<8; i++) {
            if ( ! (istringstream(argv[i]) >> posFinLin[i-5]) ) {
                std::cout << "Error: Argumento inválido. Debe ingresar un
número."<<endl;
                exit(1);
            }
        }
        //Se calcula la trayectoria desde el punto inicial hasta el final
        int returnAnglesLin[3][sample];
        linearTraj (posIniLin, posFinLin, longitudesBrazo, returnAnglesLin);
        //Se mueve el brazo a través de la trayectoria
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){
                string sal= AXServoMove(servoID[i], returnAnglesLin[i][j]);
                uartServos.write(sal);
            }
            //std::cout << "B: " << returnAnglesLin[0][j] << "  H: " <<
returnAnglesLin[1][j] << "  C: " << returnAnglesLin[2][j] << std::endl;
            usleep(twait*1000);
        }
    }

    //*** Movimiento Libre ***
    //Se mueve el brazo en una trayectoria libre
    else if (strcmp (argv[1], "-f")==0){

            std::cout << "Info: Movimiento Libre según las posiciones de entrada." <<
std::endl;
            //Revisa que se tengan al menos 6 argumentos de entrada
            if (argc < 8){
                std::cout << "Error: Argumentos insuficientes. Se necesitan 6 valores:
coordenadas del punto inicial y coordenadas del punto final" << std::endl;
                exit(1);
            }
            //Advierte si hay más argumentos de la cuenta
            if (argc > 8){
                std::cout << "Advertencia: Muchos argumentos. Se necesitan 6 valores de
posición. Se utilizarán sólo los 6 primeros argumentos." << std::endl;
            }
            //Revisa que los 6 argumentos sean números
            float posIniFree[3];
            float posFinFree[3];
            for (int i=2; i<5; i++) {
                if ( ! (istringstream(argv[i]) >> posIniFree[i-2]) ) {
                    std::cout << "Error: Argumento inválido. Debe ingresar un
número."<<endl;
                    exit(1);
                }
            }
            for (int i=5; i<8; i++) {
                if ( ! (istringstream(argv[i]) >> posFinFree[i-5]) ) {
                    std::cout << "Error: Argumento inválido. Debe ingresar un
número."<<endl;
                    exit(1);
                }
            }
            //Se calcula la trayectoria desde el punto inicial hasta el final
            int returnAnglesFree[3][sample];
            freeTraj (posIniFree, posFinFree, longitudesBrazo, returnAnglesFree);
            //Se mueve el brazo a través de la trayectoria
            for (int j=1;j<=sample;j++){
                for (int i=0; i<3; i++){
                    string sal= AXServoMove(servoID[i], returnAnglesFree[i][j]);
                    uartServos.write(sal);
                }
                //std::cout << "B: " << returnAnglesFree[0][j] << "  H: " <<
returnAnglesFree[1][j] << "  C: " << returnAnglesFree[2][j] << std::endl;
                usleep(twait*1000);
            }
    }


    //*** Demostración ***
```

```
    //Se mueve el brazo a puntos predefinidos para una demostración (generación de
letras)
    else if (strcmp (argv[1], "-d")==0){

        int returnAnglesDemo[3][sample];

        //----------LETRA H ---------------
        freeTraj (posAtaqueH1, posH1i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posH1i, posH1f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posH1f, posAtaqueH2, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posAtaqueH2, posH2i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posH2i, posH2f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posH2f, posAtaqueH3, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posAtaqueH3, posH3i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posH3i, posH3f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posH3f, posAtaqueO, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}


        //----------LETRA O ---------------
        freeTraj (posAtaqueO, posO1i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posO1i, posO1f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posO1f, posO2f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
```

```
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posO2f, posO3f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posO3f, posO1i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posO1i, posAtaqueL, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}


        //----------LETRA L ---------------
        freeTraj (posAtaqueL, posL1i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posL1i, posL1f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posL1f, posL2f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posL2f, posAtaqueA1, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}


        //----------LETRA A ---------------
        freeTraj (posAtaqueA1, posA1i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posA1i, posA1f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posA1f, posAtaqueA2, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posAtaqueA2, posA2i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posA2i, posA2f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
```

```
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posA2f, posAtaqueA3, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posAtaqueA3, posA3i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posA3i, posA3f, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posA3f, posAtaqueA1, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posAtaqueA1, posA1i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        linearTraj (posA1i, posA2i, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

        freeTraj (posA2i, posAtaqueA2, longitudesBrazo, returnAnglesDemo);
        for (int j=1;j<=sample;j++){
            for (int i=0; i<3; i++){ uartServos.write(AXServoMove(servoID[i],
returnAnglesDemo[i][j])); }
            usleep(twait*1000);}

    }
    else {
            std::cout << "Error: Parámetro de entrada desconocido." << std::endl;
    }



    uartServos.close();

    return 0;
}
```

4.2. Aplicación de Matlab para simulación (SEm_IK.m)

```
%%Es necesario tener el Robotics Toolbox instalado

close all;
clear all;

%% Definición de articulaciones según modelo de D-H
L(1)= Link( [-pi/2 0.025 0 pi/2 0], 'standard' );
L(2)= Link( [0  0  0.067  0   ], 'standard' );
L(3)= Link( [0  0  0.039  0 ], 'standard' );
Brazo = SerialLink(L,'base', transl(0,0,0.074));
Brazo.name = 'BrazoRX';
qi = [-pi/2 pi/2 0];
%qi = [0 0 0];
tiempo = [0:0.05:2]';

Brazo.plot(qi,'floorlevel', 0,'noarrow');

xh=0.09
yh=-0.128
zh=0.03
L0=0.074;
L1=0.025;
L2=0;
L3=0.067;
L4=0.039;

Base= atan2(zh,xh);
baseDeg= Base*180/pi;
baseDegRX= 150-(90-baseDeg)
baseML=Base;

s=sqrt(xh^2+zh^2)-L2;
t= -yh-L0-L1;
LRA=sqrt(s^2+t^2);
C5= (LRA^2-L3^2-L4^2)/(2*L3*L4);
Elbow= -atan2(sqrt(1-C5^2),C5);
elbowDeg=Elbow*180/pi;
elbowDegRX= 150-(elbowDeg)

gamma1= atan2(s,t);
gamma2= atan2(L4*sin(-Elbow), L3+L4*cos(-Elbow));
Shoulder= -(gamma1-gamma2);
shoulderDeg=Shoulder*180/pi;
shoulderDegRX= 150-(shoulderDeg)

q1= [(Base) (Shoulder+pi/2) Elbow];

Mov1=jtraj(qi,q1,tiempo);
Brazo.plot(Mov1);
```