E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Programación de un brazo robótico basado en Arduino y controlado remotamente.

Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Álvaro Ibáñez Mariñelarena

Ángel María Andueza Unanua

Pamplona, 19 de junio de 2018

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ÍNDICE DE CONTENIDO

1. RESUMEN Y PALABRAS CLAVE. 8

1.1. Resumen. 8

1.2. Palabras clave 8

2. OBJETIVOS 8

3. INTRODUCCIÓN 9

3.1. Robótica en general. 9

3.2. Arduino 9

3.3. Robótica en Arduino. 10

4. HARDWARE 11

4.1. Arduino UNO. 11

4.2. Tinkerkit Braccio 12

4.2.1 Descripción. 12

4.2.2 Partes del brazo robótico. 12

4.2.3 Motores 14

4.2.4 Pinza. 15

4.3. Tecnología Bluetooth. 17

4.3.1 Topologías 17

4.3.2 Acceso al medio 17

4.3.3 Modelo del paquete 18

4.3.4 Estados 18

4.4. Módulo Bluetooth. 19

4.4.1 Características del modulo. 19

4.4.2 Estándar IEE 802.15.1 19

4.4.3 Circuito. 19

4.4.4 Pines, entradas y salidas. 20

4.4.5 Modos de funcionamiento 20

4.4.6 Configuración del módulo. 23

4.5. Joystick. 25
4.6. Diagrama de conexionado. 26
5. SOFTWARE Y DESARROLLO 28
5.1. Cinemática directa. 28
5.1.1 Definición de los sistemas coordenados. 29
5.1.2 Elección de la posición casa para hallar la tabla. 29
5.1.3 Tabla de Denavit-Hartenberg. 30
5.1.4 Matriz de transformación resultante de D-H. 30
5.2. Cinemática inversa. 32
5.2.1 Introducción. 32
5.2.2 Conceptos y definiciones. 32
5.2.3 Desacoplo cinemático. 34
5.2.1 Solución de θ_1 (base o tronco). 36
5.2.2 Solución de θ_3 (codo). 37
5.2.1 Solución de θ_2 (hombro). 39
5.2.2 Solución de θ_4 (ángulo de giro vertical de la pinza). 40
5.2.3 Solución de θ_5 (ángulo de giro de la pinza). 40
5.2.4 Aplicación al Robot 41
5.2.5 Problema de la elevación 41
5.3. Mejora del posicionamiento. 43
5.3.1 Ensayo de precisión y medición del error. 43
5.3.2 Error para semiplano derecho (x>=0). 45
5.3.3 Error para semiplano izquierdo (x<0). 47
5.3.4 Algoritmo de corrección. 49
5.3.5 Perdida del área de trabajo en la corrección del error. 49
5.3.6 Problema con puntos cercanos al eje x. 50
5.3.7 Aplicación del algoritmo. 50
5.3.8 Algoritmo final. 52

5.3.9 Observaciones sobre la corrección del error. 53
5.4. Agarre de objetos con la pinza 54
5.4.1 Introducción. 54
5.4.2 Ensayo y resultados. 54
5.5. Programación en Arduino. 55
5.5.1 Introducción. 55
5.5.2 Librerías. 55
5.5.3 Variables utilizadas. 56
5.5.4 Funciones y diagramas de flujo de la programación en Arduino. 57
5.6. Sistema de creación de aplicaciones MIT App Inventor. 64
5.6.1 Necesidad de la aplicación. 66
5.6.2 Sistema MIT App Inventor. 66
5.7. Creación de la aplicación Bluetooth. 67
5.8. Comunicación Arduino con dispositivo Bluetooth. 69
5.8.1 Funciones utilizadas. 69
5.8.2 Mensaje que se envía desde el dispositivo Bluetooth. 69
5.8.3 Envío de mensaje desde Smartphone. 70
5.8.4 Recepción de mensaje en Smartphone. 70
5.8.5 Envío desde Arduino. 71
5.8.6 Recepción desde Arduino. 71
6. RESULTADOS Y DISCUSIÓN. 72
6.1. Videos de demostración del funcionamiento. 72
6.2. Discusión 72
6.3. Cumplimiento de objetivos. 74
7. CONCLUSIONES. 75
8. LINEAS FUTURAS. 76
9. BIBLIOGRAFÍA. 77

ÍNDICE DE FIGURAS

Figura 1 Robot DELTA realizando un proceso de colocación. Fuente: http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/clasificacion-de-robots/ 9

Figura 2 Robot SCARA. Fuente: http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/clasificacion-de-robots/ 9

Figura 3 Robot comercial SPC MAKEBLOCK. Fuente: http://www.homegallery.es/catalog/product/spc-makeblock-ultimate-20-kit-de-iniciacion-a-la-robotica-10-en-1 10

Figura 4 Arduino UNO. 11

Figura 5 Shield control Tinkerkit Braccio. Fuente: https://www.rs-online.com/designspark/rel-assets/ds-assets/uploads/images/5791fbff4e1c48d89daa2d300ab5c436Shield.JPG 12

Figura 6 Tinkerkit Braccio. Fuente: https://es.rs-online.com/web/p/kits-de-desarrollo-de-procesador-y-microcontrolador/1113738/ 12

Figura 7 Partes del brazo articulado. 13

Figura 8 Giro y elevación de la pinza 13

Figura 9 Imagen del servomotor SR311. 14

Figura 10 Esquemas de dimensiones del servomotor. 14

Figura 11 Imagen del servomotor SR431. 15

Figura 12 Pinza del robot con detalle de sus partes. 15

Figura 13 Pinza en posición abierta. 16

Figura 14 Pinza cerrada completamente. 16

Figura 15 Módulo bluetooth HC-05. Fuente: https://unprogramador.com/configuracion-del-modulo-bluetooh-hc-05/ 19

Figura 16 Circuito del módulo bluetooth HC05. 22

Figura 17 Módulo bluetooth. La parte trasera con los nombres de los pines. 22

Figura 18 Configuración del módulo Bluetooth. Envío y recepción en la consola COM. ... 24

Figura 19 Joystick para Arduino. Fuente: https://www.prometec.net/joystick-servo/ 25

Figura 20 Montaje de la shield en el Arduino UNO. 26

Figura 21 Diagrama de conexionado. 27

Figura 22 Cinemática directa. Colocación de sistemas coordenados mediante mecanismo de Denavit-Hartenberg 29

Figura 23 Cinemática inversa. Cálculo de $\theta 1$. 36

Figura 24 Cinemática inversa. Cálculo de $\theta 3$. 37

Figura 25 Cinemática inversa. Cálculo de $\theta 2$. 39

Figura 26 Cinemática inversa. Cálculo de theta4. 40

Figura 27 Plano con robot colocado. 43

Figura 28 Forma de medir el error. 44

Figura 29 Robot alcanzando un punto en el plano. 45

Figura 30 Medición del error en semiplano derecho. Error_x (medio) con respecto a x. .. 46

Figura 31 Medición del error en semiplano derecho. Error y (medio) respecto a x. 47

Figura 32 Medición del error en semiplano izquierdo. Error x (medio) respecto x. 48

Figura 33 Medición del error en semiplano izquierdo. Error y (medio) respecto x. 48

Figura 34 Aplicación del algoritmo. Error_x en función de x e y. 50

Figura 35 Aplicación del algoritmo. Error_y en función de x e y. 51

Figura 36 Gráfico ángulo de la pinza frente diámetro del objeto agarrado. 54

Figura 37 Diagrama de flujo de la función que calcula la cinemática inversa. 58

Figura 38 Diagrama de flujo de la función que mueve el robot a unos ángulos de los servomotores. 59

Figura 39 Diagrama de flujo de la función dadas unas coordenadas xyz, mueve el robot a dicho punto. 59

Figura 40 Diagrama de flujo de la función que aplica un ángulo al servo de la pinza. 60

Figura 41 Diagrama de flujo de la función que abre la pinza a una distancia dada..... 60

Figura 42 Diagrama de flujo de la función que mueve el robot a la posición casa. 61

Figura 43 Diagrama de flujo de la función de ajuste de la velocidad de movimiento. 61

Figura 44 Diagrama de flujo de la función que almacena las variables articulares de un punto en la matriz. 62

Figura 45 Diagrama de flujo de la función que recoge el punto guardado en la matriz en un vector y un entero. 62

Figura 46 Diagrama de flujo de la función que borra un punto (fila) de la matriz. 63

Figura 47 Diagrama de flujo de la función que calcula las coordenadas de la posición actual en un vector. 63

Figura 48 Ejemplo de bloque de programación de MIT App Inventor. 66

Figura 49 App para control del robot en smartphone. Pantalla cuando está desconectado. 67

Figura 50 App para control del robot en smartphone. Pantalla cuando está conectado y en control con coordenadas xyz 67

Figura 51 App para control del robot en smartphone. Pantalla cuando está conectado y en control con coordenadas xyz 68

Figura 52 App para control del robot en smartphone. Pantalla cuando está conectado y en control con ángulos. 68

Figura 53 Detalle de la pieza de unión. 73

Figura 54 Conjunto eslabón base con servomotor. 73

1. RESUMEN Y PALABRAS CLAVE.

1.1. Resumen.

El presente Proyecto consiste en la programación de un brazo robótico utilizando la plataforma de programación de Arduino. El desarrollo del proyecto consistirá en la programación del brazo robótico comercial Tinkerkit Braccio empleando el software propio de Arduino, así como en la búsqueda de estrategias de mejora de la precisión en su movimiento. Además, se pretende diseñar una aplicación para el control vía el uso de un smartphone o de otros periféricos como un joystick.

1.2. Palabras clave.

- Robótica
- Arduino
- Bluetooth
- Android

2. OBJETIVOS

El objetivo del presente proyecto es obtener un control de un brazo articulado mediante el uso de Arduino. Este control se hará mediante un método cómodo para el usuario utilizando dispositivos Android.

Se busca aplicar los conocimientos sobre la robótica junto a la programación en Arduino y desarrollar un programa con estos objetivos:

- Control posicional preciso del brazo destinado al agarre y colocación de objetos. El posicionamiento se realizará mediante coordenadas en el espacio xyz.
- Implementación de algoritmos de corrección del error y mejora de la precisión, así como el agarre óptimo de objetos con la pinza.
- Diseño de una interfaz de control sencilla de utilizar y que se use desde PC o desde smartphone de forma inalámbrica. También podrá hacerse un control manual con un joystick.

3. INTRODUCCIÓN

3.1. Robótica en general.

La robótica es la ciencia que estudia el diseño y aplicaciones de los robots. Los robots son máquinas programables capaces de realizar movimientos en una aplicación concreta. Es un área multidisciplinar ya que requiere conocimientos de mecánica, electrónica y programación.

Existen múltiples tipos de robots como pueden ser: robots industriales, robots de servicio, de investigación, robots médicos y militares entre otros. Sus características pueden diferir enormemente de un tipo a otro, aunque tienen características en común. La reprogramación, el control a distancia, la capacidad de procesamiento y adaptación al entorno son cualidades que deben existir en cuenta en todos los robots.

Hoy en día, la aplicación de la robótica es muy común en la industria para tareas de manipulación y de procesamiento dada su capacidad de trabajar en ambientes peligrosos o nocivos para el humano. Las nuevas tendencias sobre la robótica como Internet of Robots y Robótica Colaborativa.

Figura 2 Robot SCARA. Fuente: http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/clasificacion-de-robots/

Figura 1 Robot DELTA realizando un proceso de colocación. Fuente: http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/clasificacion-de-robots/

3.2. Arduino

Arduino es una plataforma de hardware abierto y software libre basada en el uso de un microcontrolador. Toma y une características de las plataformas Wiring y Processing.

Wiring: Plataforma que permite el control de microcontroladores mediante lenguaje de programación. Es una plataforma pensada para programadores.

Processing: Es un entorno de desarrollo de código abierto. Permite a no programadores crear proyectos gracias a la aportación de librerías y ejemplos disponibles en su página web.

De Wiring toma el lenguaje C/C++ simplificado junto al compilador. De Processing coge el entorno de desarrollo con la utilización de librerías y una web donde se comparten los proyectos.

Plataforma de hardware abierto: agrupa el conjunto de dispositivos de hardware cuyos diagramas, layouts, etc. son de acceso público (que puede ser de pago o gratuito).

El desarrollo del hardware libre tiene ventajas como la reutilización de diseños para la innovación y ahorro de costes. Sin embargo, presenta desafíos como la dificultad en la producción y la capacidad de reproducción del diseño. Otros ejemplos de hardware abierto se encuentran en las Impresoras 3D, cámaras configurables y teléfonos.

El hecho de que Arduino sea una plataforma de open hardware ha permitido la realización de múltiples proyectos de robótica, impresión 3D e internet de las cosas entre otros. La presencia de una gran comunidad con múltiples cursos y tutoriales, herramientas de consulta y proyectos realizados por otras personas permite el aprendizaje de una forma sencilla.

3.3. Robótica en Arduino.

Existen cientos de proyectos de robótica realizados por la comunidad en la plataforma Arduino. Las temáticas de estos proyectos van desde la fabricación del robot hasta la programación de un robot comercial en una aplicación determinada. Existen soluciones comerciales de bajo coste para iniciarse en la robótica.

El entorno de programación, el hecho de poder consultar proyectos realizados con anterioridad y la gran variedad de módulos disponibles para Arduino permite múltiples aplicaciones como robots móviles, robots con cámara integrada, etc.

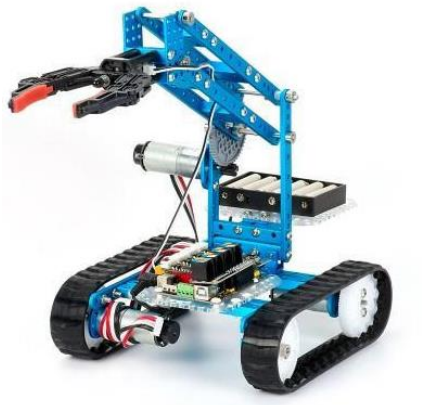


Figura 3 Robot comercial SPC MAKEBLOCK. Fuente: http://www.homegallery.es/catalog/product/spc-makeblock-ultimate-20-kit-de-iniciacion-a-la-robotica-10-en-1

4. HARDWARE

En este apartado se describirán los elementos de hardware utilizados.

4.1. Arduino UNO.

Se utiliza esta placa debido a la compatibilidad con el módulo shield del brazo robótico y también cuenta con pines suficientes para esta aplicación.

Es una placa basada en el microcontrolador ATmega328. Cuenta con:

- 14 salidas digitales.
- 6 entradas analógicas.
- Conector a alimentación.
- Conector a ordenador USB.
- Botón de reset.

Figura 4 Arduino UNO.

4.2. Tinkerkit Braccio

4.2.1 Descripción.

El Tinkerkit Braccio es un brazo robótico comercial diseñado para su utilización con Arduino. Utiliza 6 servomotores que se controlan a través de un shield que se conecta directamente a la placa Arduino UNO.

Se trata de un brazo robótico de 5 grados de libertad con una pinza diseñada para el agarre de objetos. La fabricación de la pinza como sus eslabones es en plástico rígido.

Figura 6 Tinkerkit Braccio. Fuente: https://es.rs-online.com/web/p/kits-de-desarrollo-de-procesador-y-microcontrolador/1113738/

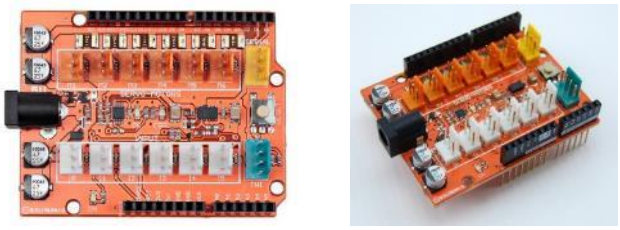


Figura 5 Shield control Tinkerkit Braccio. Fuente: https://www.rs-online.com/designspark/rel-assets/ds-assets/uploads/images/5791fbff4e1c48d89daa2d300ab5c436Shield.JPG

4.2.2 Partes del brazo robótico.

El brazo robótico consta de 6 eslabones con movimientos relativos de giro en un eje entre ellos. Su construcción y funcionamiento se asemeja a la de un brazo humano (robot antropomórfico), aunque sustituyendo la mano por una pinza y restando un giro de la muñeca. Para nombrar cada una de sus partes se aprovecha su correspondencia a las partes del brazo.

A continuación, se nombran las partes del brazo articulado (en la Figura 7) a las que luego se hará referencia en apartados posteriores.

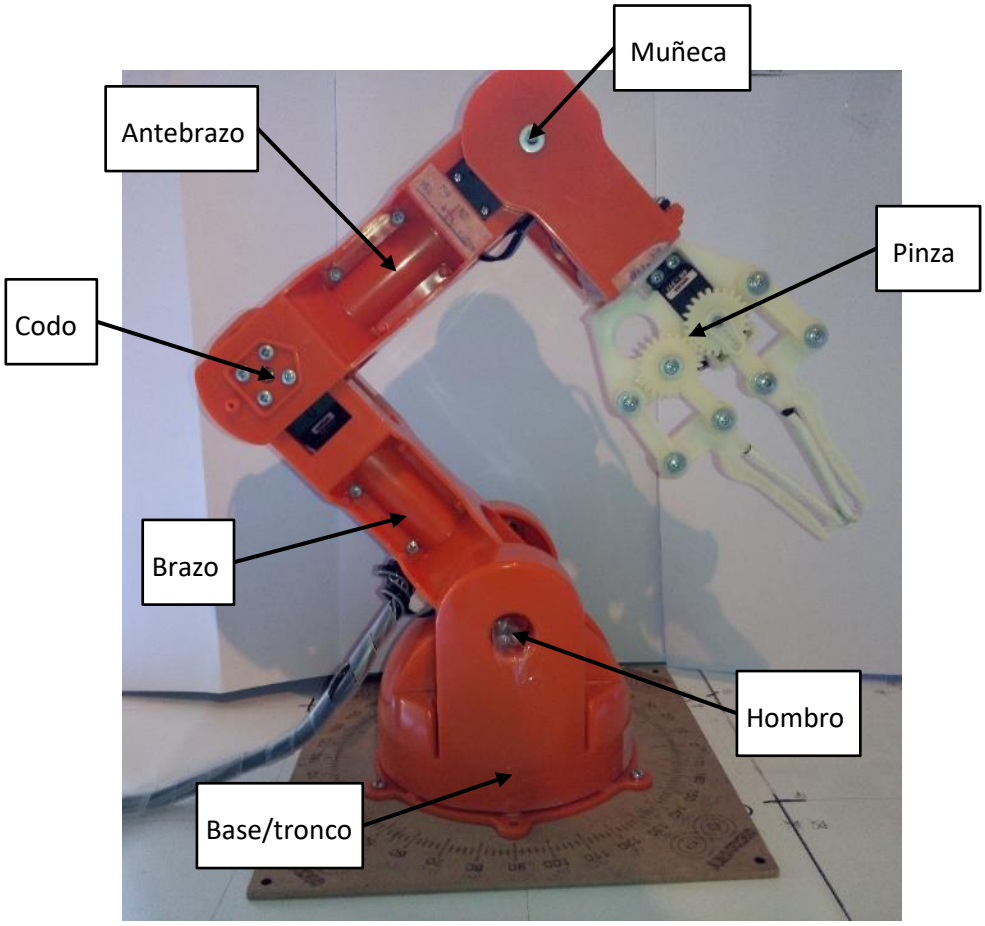


Figura 7 Partes del brazo articulado.

A diferencia de las demás articulaciones, en la muñeca se dan dos giros.

- **Elevación de la pinza:** es el movimiento de giro del conjunto pinza respecto al antebrazo con el que se consigue variar el ángulo que forman entre ellos.
- **Giro de la pinza:** va después de la elevación de la pinza en la cadena cinemática por lo que no altera el ángulo de la pinza respecto al antebrazo.

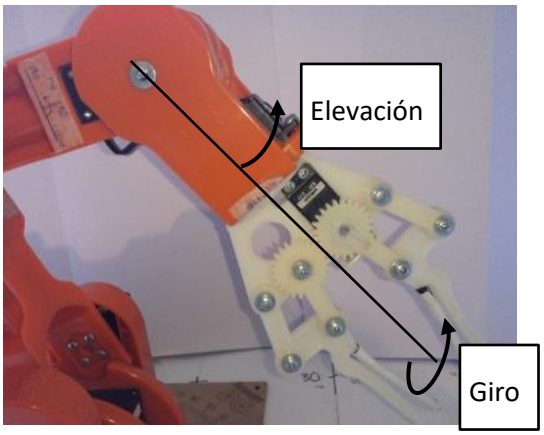


Figura 8 Giro y elevación de la pinza.

4.2.3 Motores.

El brazo robótico utiliza servomotores como accionamiento. Un servo motor es un motor de corriente continua que es capaz de ubicarse en cualquier posición de su rango de operación y mantenerse estable en dicha posición. Su control se hace mediante una señal PWM de 20 microsegundos en la que se varía el ciclo de trabajo.

Cada articulación del brazo lleva asociado un servomotor. Hay dos tipos de servomotores utilizados en este dispositivo.

SpringRC SR311

Se encuentran dos servos de este tipo en el robot. Uno en la articulación del giro de la muñeca y otro en el giro de la pinza. Son de pequeño peso y tamaño reducido.

Figura 9 Imagen del servomotor SR311.

Especificaciones del producto							Parámetros técnicos				
Tamaño (mm)					Peso (g)	Cable (cm)	**4.8V**		**6V**		Ángulo de rotación (°)
A	B	C	D	E			Velocidad (s/60°)	Torque (kg*cm)	Velocidad (s/60°)	Torque (kg*cm)	
31.3	16.5	28.6	38.1	8	27	20	0.14	3.1	0.12	3.8	180

Tabla 1 Especificaciones del servomotor SR311.

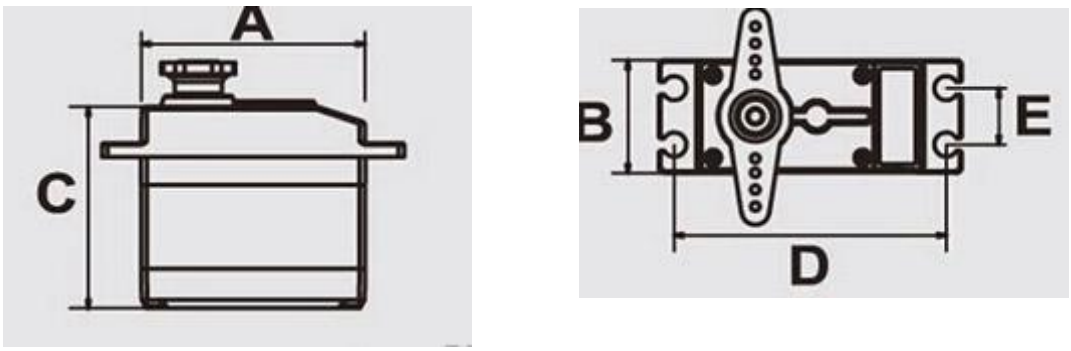


Figura 10 Esquemas de dimensiones del servomotor.

SpringRC SR431

Es un servomotor de tamaño mediano y alto par. Hay cuatro servos de este tipo en el robot colocados en las articulaciones que más fuerza requieren como son el tronco, el hombro, el codo y la muñeca de elevación.

Figura 11 Imagen del servomotor SR431.

Especificaciones del producto							Parámetros técnicos				
Tamaño (mm)					Peso (g)	Cable (cm)	**4.8V**		**6V**		Ángulo de rotación (°)
A	B	C	D	E			Velocidad (s/60°)	Torque (kg*cm)	Velocidad (s/60°)	Torque (kg*cm)	
42	20.5	39.5	49	10	62	30	0.2	12.2	0.18	14.5	180

Tabla 2 Especificaciones servomotor SR431.

4.2.4 Pinza.

El fin de este robot es coger y desplazar objetos con la pinza. En este apartado se va a describir las características de la pinza, sus problemas.

Partes de la pinza

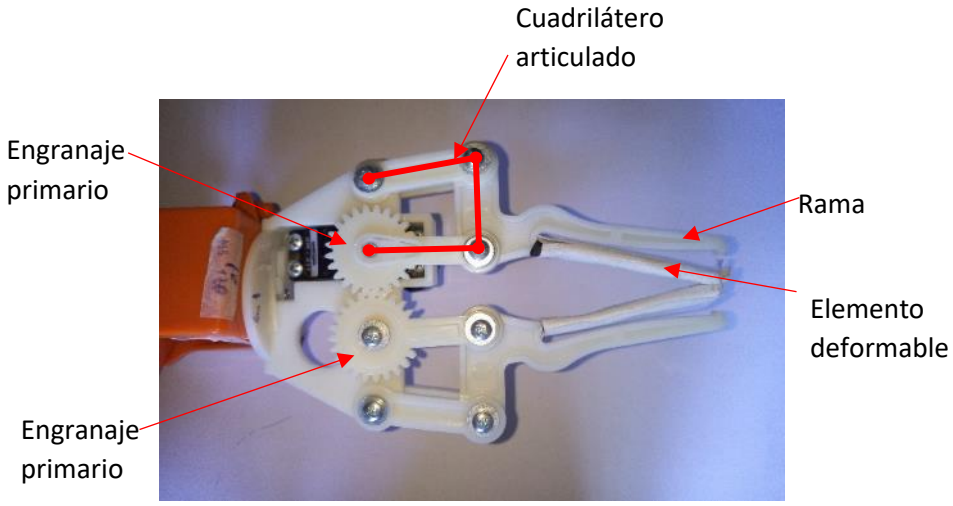


Figura 12 Pinza del robot con detalle de sus partes.

Descripción de la pinza

El mecanismo de la pinza es un cuadrilátero articulado donde una de sus barras está gira sobre su base mediante el servomotor. Ambas barras miden aproximadamente lo mismo por lo que la barra superior se desplaza sin variar su ángulo respecto al robot. Se consigue un desplazamiento de su rama (perpendicular a la barra superior) con un ángulo casi constante.

Mediante dos engranajes del mismo número de dientes, uno unido al servomotor y otro unido a otro cuadrilátero articulado se consigue el movimiento simétrico de la otra rama de la pinza. Así se logra el movimiento de agarre.

Figura 13 Pinza en posición abierta.

Figura 14 Pinza cerrada completamente.

A parte cuenta con unos elementos elásticos que se deforman acomodándose al objeto a agarrar. Esto tenía una serie de problemas:

- Dificultad para agarrar objetos de poco diámetro ya que necesita que la pinza se cierre más que el diámetro del objeto.
- La superficie de contacto con el objeto era pequeña e imposibilitaba el agarre de objetos grandes y pesados.
- Otro problema era que los objetos se resbalaban, la solución fue colocar cinta adhesiva rugosa para mejorar el agarre.

4.3. Tecnología Bluetooth.

Para la implementación de control remoto en brazo articulado es necesario un protocolo de comunicaciones inalámbrico. La tecnología bluetooth se adapta bien ya que se basa en redes personales (de corto alcance), la mayor parte de dispositivos móviles cuentan con bluetooth y hay módulos disponibles para dotar de esta funcionalidad al microcontrolador Arduino.

4.3.1 Topologías

En las comunicaciones bluetooth estas son las topologías de redes utilizadas. Los elementos de estas redes pueden ser maestros o esclavos.

Piconets: constan de cómo máximo 7 elementos. Solo un maestro y los demás esclavos. Un elemento de una piconet solo ejecutará las órdenes recibidas por su maestro, ignorando las demás.

Scatternets: cuando dos dispositivos pertenecientes a dos piconets diferentes se conectan se forman las scatternets. Son una extensión de la red que soporta hasta 10 piconets interconectadas.

4.3.2 Acceso al medio

El principal problema de las comunicaciones inalámbricas son las interferencias, sobre todo en entorno donde hay más redes bluetooth o de otro tipo. Para solventar este problema se utiliza una estrategia de **"saltos de frecuencia"**.

Se va alternando la frecuencia de comunicación entre 77 niveles disponibles a una distancia de 1MHz cada uno. Estos niveles se sitúan en torno a 2.4GHz para ser reconocibles en la comunicación. De este modo se consigue diferenciar entre mensajes de diferentes redes simplemente escogiendo la frecuencia adecuada.

El envío de datos entre los dispositivos se hace mediante una **multiplexación de los canales utilizando división en el tiempo (TDM)**. Los paquetes de datos se organizan en tramas compuestas por 1, 3 ó 5 ranuras temporales. Una ranura temporal es una subdivisión de la trama dedicada a que un dispositivo envíe un dato (por ejemplo, un bit o un carácter), en la siguiente ranura será otro dispositivo el que envíe otro dato.

El método de acceso al medio es **CDMA (acceso múltiple por división de código)** basado en la tecnología de espectro expandido que es muy eficiente ya que les afecta menos ruido cuando coexiste con otras señales.

Para el control de acceso se utiliza la **técnica de reagrupación**. El maestro transmite en una ranura e indica qué esclavo transmite en la siguiente.

4.3.3 Modelo del paquete

El paquete se compone de 3 partes: Código de acceso, cabecera y datos.

Dirección	Tipo	Datos
72 bits	54 bits	De 0 a 2745 bits

Código de acceso

Puede ser de 3 tipos:

- Channel Access Code (CAC): identifica la piconet.
- Device Access Code (DAC): se usa durante el pedido de conexión.
- Inquiry Access Code (IAC): identifica los dispositivos de la red que son operativos y sus direcciones.

Cabecera

Dirección	Tipo	Control de flujo	ACK/NACK	Número de secuencia	Control de errores
3 bits	4 bits	1 bit	1bit	1bit	8bits

Datos

Encabezado	Cuerpo	Código CRC
De 8 a 16 bits	Tamaño variable	16 bits

4.3.4 Estados

Desconectado: antes de la conexión de una piconet el dispositivo está a la espera de pedidos de conexión.

Conectado: participa en el tráfico de datos. Si participa intensamente está en modo activo y si no, puede estar en modo bajo consumo.

4.4. Módulo Bluetooth.

4.4.1 Características del módulo.

Trabaja con el Puerto serie USART de Arduino (pines 0 y 1).

Sigue el IEEE 802.15.1 standardized protocol.

4.4.2 Estándar IEE 802.15.1

Define la capa física y de acceso al medio, niveles 1 y 2 de la pila OSI. Se usa en redes de entorno personal o WPAN.

Permite el envío de bits, pero no sirve para el envío de multimedia (imágenes y videos).

4.4.3 Circuito.

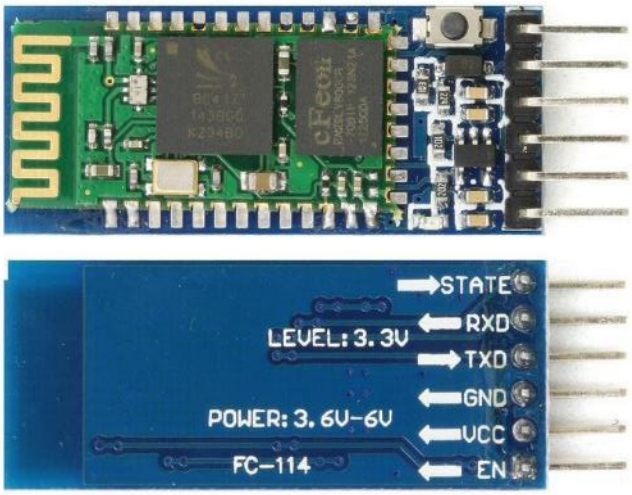


Figura 15 Módulo bluetooth HC-05. Fuente: https://unprogramador.com/configuracion-del-modulo-bluetooh-hc-05/

En el circuito se pueden distinguir dos partes:

- El módulo bluetooth HC05.
- Circuito de adaptación de tensión y pines macho.

El circuito de adaptación de la tensión de 5V a 3.3V. Se compone de resistencias y un regulador lineal. La salida de 3.3V se utiliza tanto para alimentar al módulo bluetooth como para el LED indicador del estado.

4.4.4 Pines, entradas y salidas.

- 5V → Alimentación
- GND → Masa
- TX → Pin de transmisión. Se conecta al pin de recepción de Arduino.
- RX → Pin de recepción. Se conecta al pin de transmisión de Arduino.
- EN → Sirve para entrar en modo configuración. En este modelo no se utiliza ya que cuenta con un botón
- STATE → Se pone a 1 cuando se ha conectado a otro dispositivo (se forma la piconet). No se utilizará en nuestro diseño.

Se puede ver el orden de los pines en la Figura 17.

4.4.5 Modos de funcionamiento

A continuación, se describirán los modos en los que puede funcionar este módulo bluetooth en concreto.

- Estado desconectado.
- Estado conectado.
- Modo AT 1.
- Modo AT 2.

Estado desconectado

Entra en este estado al alimentar el módulo y mientras no se ha establecido conexión con ningún otro dispositivo. El LED parpadea rápidamente. No interpreta comandos AT.

Estado conectado

Entra cuando se establece conexión con otro dispositivo bluetooth.

El LED hace un doble parpadeo.

LA comunicación es transparente, es decir, todo dato que llegue al módulo por el pin RX se transmite por bluetooth y todo dato que llegue por bluetooth se transmite por el pin TX.

Modo AT 1

Se pueden enviar comandos AT, pero a la velocidad de configuración (en baudios). Algunos comandos AT no están disponibles en este modo.

Para entrar en este estado hay que conectar el PIN 34 a 3.3V después de alimentar el módulo.

El LED parpadea rápidamente como en el modo desconectado.

Modo AT 2

Para entrar en este modo es necesario conectar 3.3V al PIN 34 antes de alimentar al módulo. Una vez alimentado permanecerá en este estado, aunque se cambie ese nivel tensión.

Con este módulo se ha tenido que hacer llevando un cable directamente debido a la problemática del circuito del botón. En el modo de configuración el LED tiene que hacer un parpadeo lento.

En este modo interpreta todos los comandos AT. Algunos comandos AT solo los interpreta si hay 3.3V conectados al PIN34.

La velocidad para enviar comandos AT es de 38400 baudios.

El LED parpadea lentamente.

Problema para entrar en el modo de configuración.

En teoría este tipo de módulos están diseñados para que al pulsar el botón una tensión de 3.3V aparezca en el PIN34, todo ello sin necesidad de añadir una tensión externa en el pin EN. En el módulo que se compró faltaba una resistencia entre el PIN34 y la salida del botón, entonces el PIN34 no estaba conectado al circuito. Por esta razón el botón pierde toda la funcionalidad.

Solución:

Ya que en esta aplicación solo hay que entrar en el modo de configuración una vez para configurar el módulo, se aplicará directamente la tensión de 3.3V de un pin de Arduino haciendo contacto con un cable jumper macho. Todo esto con precaución para no hacer contacto con otras partes del circuito y evitar sobrecalentamientos o cortocircuitos.

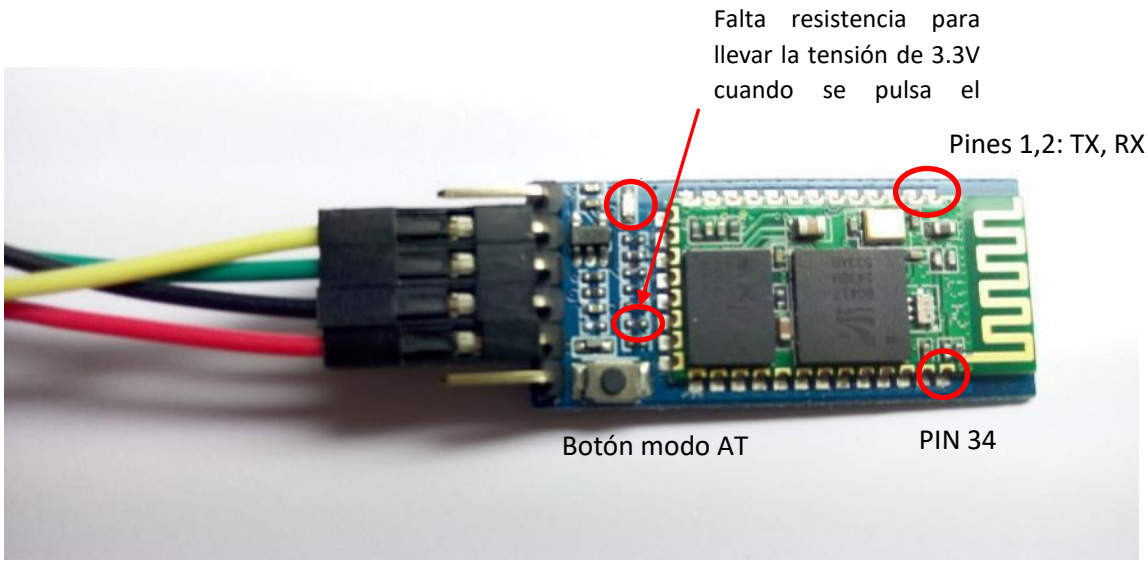


Figura 16 Circuito del módulo bluetooth HC05.

Figura 17 Módulo bluetooth. La parte trasera con los nombres de los pines.

4.4.6 Configuración del módulo.

En este apartado se describirán los pasos tomados a la hora de configurar este módulo y se mostrará el programa de Arduino utilizado para su configuración.

Durante la configuración el módulo deberá estar en el **Modo AT2**. Para entrar en este modo de configuración se debe alimentar el PIN34 del módulo a 3.3V hasta después de haber alimentado el módulo. El LED parpadeará lentamente.

La configuración del módulo se realiza enviándole comandos AT.

Lista de comandos AT utilizados

Comando AT	Respuesta	Descripción
AT	OK	Comprobación de que está funcionando
AT+RESET	OK	Salir del modo de configuración AT2 y pasar al modo desconectado.
AT+ROLE	<Rol> OK	Rol del dispositivo que tendrá durante la conexión. <Rol>= 0 → esclavo 1 → maestro
AT+ROLE=<Rol>	OK	
AT+UART	+UART:<Baud rate>,<Stop bit>,<Parity bit> OK	Devuelve el baudrate, el nº de bits de parada y el bit de paridad. <Baud rate> <Stop bit> 0 → 1 bit de parada 1 → 2 bits de parada <Parity bit>
AT+UART=<Baud rate>, <Stop bit>,<Parity bit>	OK	
AT+NAME	+NAME:<Name> OK	Requiere 3.3V en PIN 34. Nombre del dispositivo para buscar durante la conexión. <Name>
AT+ROLE=<Name>	OK	
AT+PSWD	+PSWD:<Password> OK	Contraseña del dispositivo para vincularlo con otro <Password>
AT+PSWD=<Password>	OK	

Tabla 3 Lista de comandos AT.

Lista completa de comandos AT:
https://halckemy.s3.amazonaws.com/uploads/attachments/356625/hc-05_at_commands_qqjP15d3k3.pdf

Envío de los comandos AT

El módulo bluetooth está conectado al puerto Serial de Arduino. Para enviarle mensajes basta con escribir en el puerto los caracteres a enviar, esto se realiza desde la consola COM.

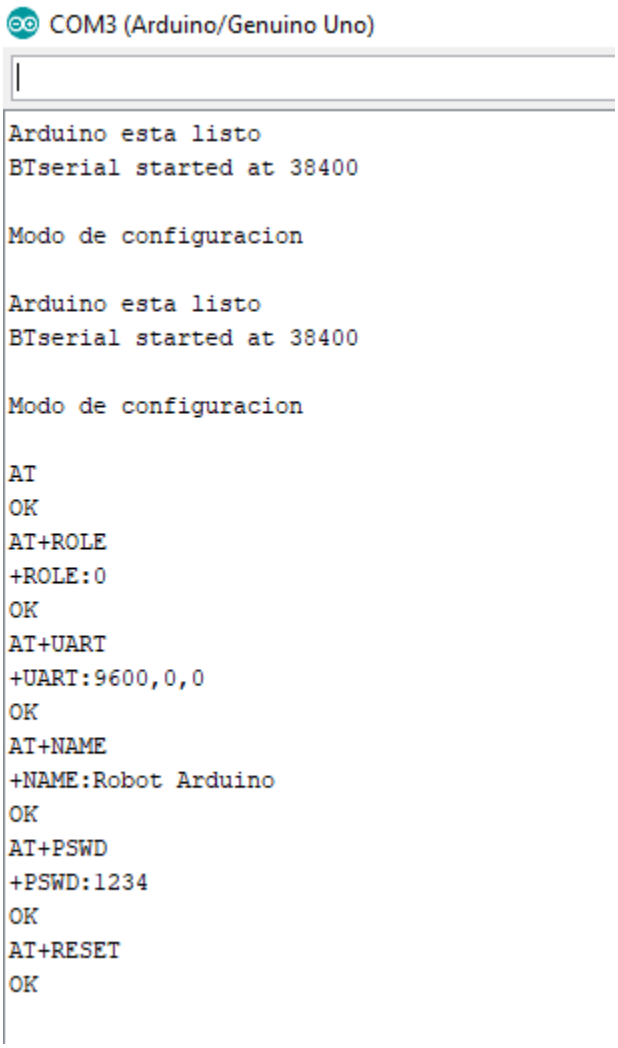


Figura 18 Configuración del módulo Bluetooth. Envío y recepción en la consola COM.

Tras comprobar que todos los parámetros son los deseados y modificar si hiciera falta, se procede al reseteo mediante el comando AT+RESET. Ahora, el módulo está listo para conectarse a otro dispositivo, el LED debe parpadear rápido.

El dispositivo por conectar debe estar vinculado con la contraseña configurada (por defecto es 0000 ó 1234).

La configuración está realizada, ya está listo para vincular con otros dispositivos como un smartphone.

4.5. Joystick.

Un joystick consiste en 2 potenciómetros que varían la señal de tensión desde 0V a 5V al mover el elemento de agarre entre sus posiciones límite. Estas dos señales de tensión son Vx y Vy y están disponibles para su lectura en 2 pines del joystick.

Estas tensiones proporcionadas son de aproximadamente 2.5V cuando está en reposo. Será necesario implementar por software que el programa no reaccione hasta que se hayan sobrepasado un límite superior o un límite inferior. Estos límites dependerán de las características del joystick, pero se pueden observar si se hacen lecturas continuadas de este valor y se muestran en el monitor Serial mediante un programa de Arduino.

A parte cuenta con un botón asociado a un pin (SW). Al presionar el joystick aparecerá una señal de 0V en el pin y será 5V cuando no se presiones

Arduino solo tiene un pin de 5V disponible que se destina a la alimentación del módulo bluetooth. Para alimentar el joystick se usa el pin de 3.3V. Esto implica que la tensión analógica de salida de los dos potenciómetros varía ahora de 0 a 3.3V (hay una pérdida de sensibilidad), así que el programa debe estar adaptado a este cambio. Este cambio no afecta al botón ya que se conecta a modo de resistencia de pull-up.

Figura 19 Joystick para Arduino. Fuente: https://www.prometec.net/joystick-servo/

Pines del joystick

Pin	Descripción
GND	Masa
5V	Alimentación 5V
VRx	Tensión en función del movimiento en x
VRy	Tensión en función del movimiento en y.
SW	Salida digital del botón

Tabla 4 Pines del joystick.

4.6. Diagrama de conexionado.

La shield que controla los motores va encajada en los pines del Arduino como se muestra en la Figura 20.

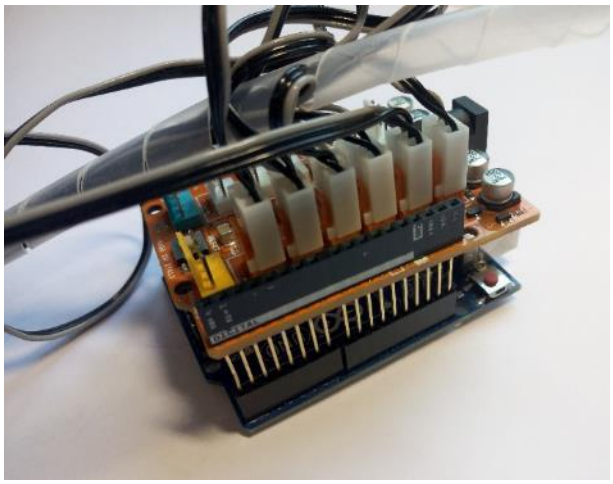


Figura 20 Montaje de la shield en el Arduino UNO.

Conexionado del módulo bluetooth HC-05.

Pin del módulo	Pin de la placa	Color de cable
STATE	No conectado	
RX	1 (TX)	
TX	0 (RX)	
GND	GND	
5V	5V	
EN	No conectado	

Tabla 5 Conexiones del módulo bluetooth a Arduino.

Conexionado del joystick.

Pin del módulo	Pin de la placa	Color de cable
GND	GND	
Vcc	3.3V	
VRx	A0	
VRy	A1	
SW	4	

Tabla 6 Conexiones del joystick a Arduino.

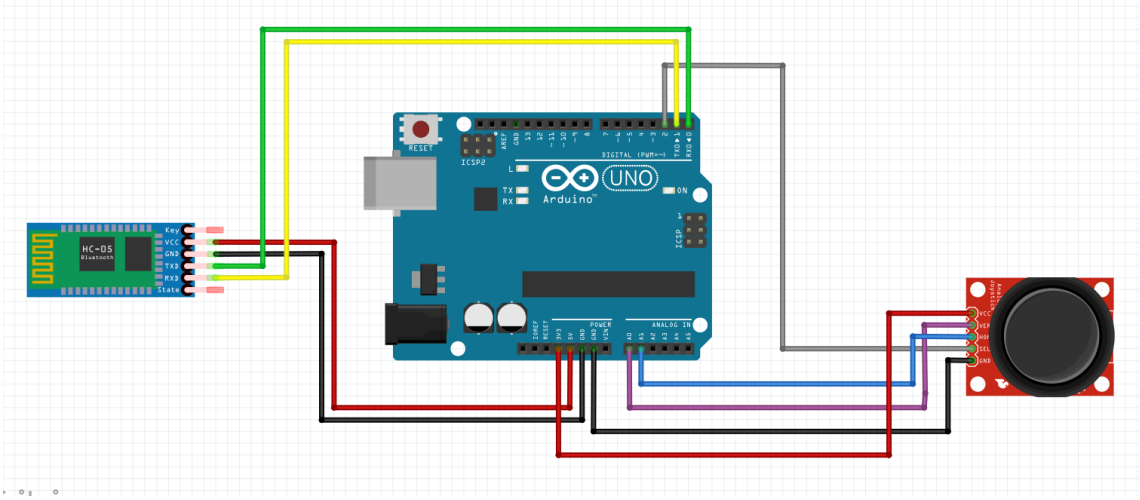


Figura 21 Diagrama de conexionado.

5. SOFTWARE Y DESARROLLO

5.1. Introducción.

En este apartado se hablará sobre el desarrollo del proyecto y la programación del brazo robótico.

El primer paso, antes de empezar a programar en Arduino, es el estudio de su cinemática o el estudio conjunto de todos sus movimientos y estudios sobre sus características relacionadas (como la precisión). Los conocimientos extraídos de la cinemática se aplicarán en su programación para conseguir el control adecuado.

Posteriormente, se tratará la programación en Arduino o en otras plataformas y la inclusión de los módulos. Básicamente se tratarán los métodos de control del brazo robótico.

5.2. Cinemática directa.

La cinemática directa consiste en asignar un sistema coordenado a cada eslabón y establecer su posición y orientación respecto a los sistemas coordenados de los eslabones adyacentes. Este apartado es necesario para caracterizar el robot y entender su comportamiento a la hora de realizar movimientos, además se definen las variables articulares (ángulos) y los sistemas coordenados a los que se hará alusión en siguientes apartados.

El objetivo es que dado el conjunto de variables articulares puedan obtenerse las posiciones y orientaciones de todos los sistemas coordenados que se han definido. Como puede observarse esta relación es unívoca ya que un conjunto de variables articulares solo puede dar lugar a una única pose.

Para la definición de los sistemas coordenados se utilizará el procedimiento de Denavit-Hartenberg, que es un conjunto de reglas para definirlos de una forma conveniente. El resultado se plasmará en forma de tabla y ya podrán calcularse las matrices de transformación homogéneas. (Ollero, A. (2001). *Robótica: manipuladores y robots móviles.* (pp. 66-73). Universidad de Sevilla.)

5.2.1 Definición de los sistemas coordenados.

Aplicando el método de Denavit-Hartenberg para el robot se han deducido los sistemas coordenados representados en la siguiente figura.

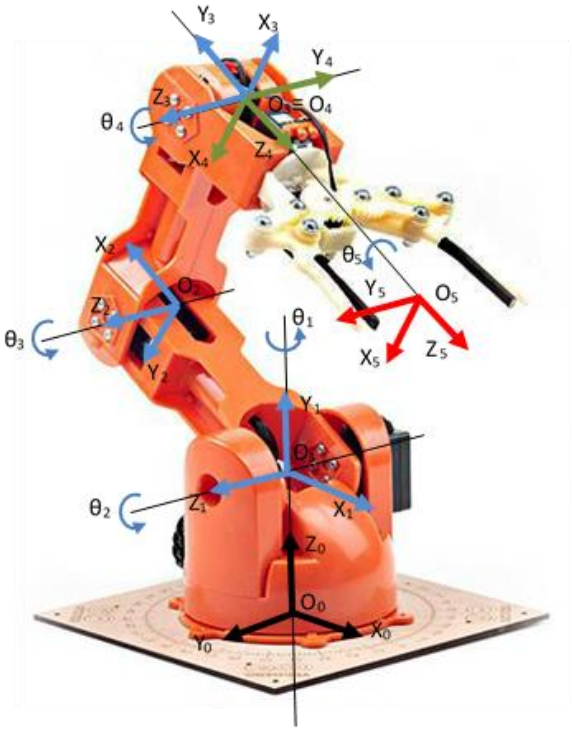


Figura 22 Cinemática directa. Colocación de sistemas coordenados mediante mecanismo de Denavit-Hartenberg

5.2.2 Elección de la posición casa para hallar la tabla.

Se escoge la posición casa como la posición de inicio del robot, pero con el ángulo de tronco a 90° para dar más libertad a la hora de desplazarse a ambos lados sin que llegue el servo a su tope.

La posición casa se ha elegido con estás ángulos de los servomotores (en grados sexagesimales):

Servo TRONCO = 90

Servo HOMBRO = 45

Servo CODO = 180

Servo MUÑECA_ELEV = 90

Servo MUÑECA_GIRO = 0

Entonces, los offsets articulares (en grados sexagesimales) quedan:

$\theta_{10} = 0$

$\theta_{20} = 145$

$\theta_{30} = -90$

$\theta_{40} = 180$

$\theta_{50} = 0$

5.2.3 Tabla de Denavit-Hartenberg.

	i	θ_i+offset$_i$	d_i	a_i	α_i
0→1	1	θ_1+0°	L1	0	90°
1→2	2	θ_2+145°	0	L2	0°
2→3	3	θ_3-90°	0	L3	0°
3→4	4	θ_4+180°	0	0	-90°
4→5	5	θ_5+0°	L5	0	0°

Tabla 7 Tabla de Denavit-Hartenberg.

Con motivo de abreviar durante el uso de operaciones coseno y seno con estos ángulos se denotará:

$\cos(\theta_i + \text{offset}_i) = c_i$ $\quad$ $\text{sen}(\theta_i + \text{offset}_i) = s_i$

Longitud de los eslabones

L1=6.5cm L2=12.5cm

L3=12.5cm L5=19cm

5.2.4 Matriz de transformación resultante de D-H.

Utilizando la tabla se pueden hallar las matrices de transformación de los sistemas coordenados y conseguir la orientación y distancia del centro de la pinza respecto el origen.

Representación de una matriz de transformación homogéneas.

$$A_{i\to j} = \begin{bmatrix} R_{i\to j} & d_{ij} \\ 0_{1x3} & 1 \end{bmatrix} \ (Matriz\ 4x4)$$

Siendo:

d_{ij} : vector distancia entre orígenes en la base i. Vector 3x1.

$R_{i\to j}$: matriz de rotación i a j. Matriz 3x3.

Matrices de transformación i-1→i

$$A_{0\to 1} = \begin{bmatrix} c1 & 0 & s1 & 0 \\ s1 & 0 & -c1 & 0 \\ 0 & 1 & 0 & L1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_{1\to 2} = \begin{bmatrix} c2 & -s2 & 0 & L2 \\ s2 & c2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{2\to3} = \begin{bmatrix} c3 & -s3 & 0 & L3 \\ s3 & c3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_{3\to4} = \begin{bmatrix} c4 & 0 & -s4 & 0 \\ s4 & 0 & c4 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_{4\to5} = \begin{bmatrix} c5 & -s5 & 0 & 0 \\ s5 & c5 & 0 & 0 \\ 0 & 0 & 1 & L5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz de transformación final.

$$A_{0\to5} = \prod_{i=1}^{5} A_{i-1\to i}$$

Utilizando MATLAB:

$$A_{4\to5}[:,1] = \begin{bmatrix} c5(c4(c1*c2*c3 - c1*s2*s3) - s4(c1*c2*s3 + c1*c3*s2)) - s1*s5 \\ c5(c4(c2*c3*s1 - s1*s2*s3) - s4(c2*s1*s3 + c3*s1*s2)) + c1*s5 \\ c5(c4(c2*s3 + c3*s2) + s4(c2*c3 - s2*s3)) \\ 0 \end{bmatrix}$$

$$A_{4\to5}[:,2] = \begin{bmatrix} -c5*s1 - s5(c4(c1*c2*c3 - c1*s2*s3) - s4(c1*c2*s3 + c1*c3*s2)) \\ c1*c5 - s5(c4(c2*c3*s1 - s1*s2*s3) - s4(c2*s1*s3 + c3*s1*s2)) \\ -s5(c4(c2*s3 + c3*s2) + s4(c2*c3 - s2*s3)) \\ 0 \end{bmatrix}$$

$$A_{4\to5}[:,3] = \begin{bmatrix} -c4(c1*c2*s3 + c1*c3*s2) - s4(c1*c2*c3 - c1*s2*s3) \\ -c4(c2*s1*s3 + c3*s1*s2) - s4(c2*c3*s1 - s1*s2*s3) \\ c4(c2*c3 - s2*s3) - s4(c2*s3 + c3*s2) \\ 0 \end{bmatrix}$$

$$A_{4\to5}[:,4] = \begin{bmatrix} L2*c1 - L5(c4(c1*c2*s3 + c1*c3*s2) + s4(c1*c2*c3 - c1*s2*s3)) + L3*c1*c2 \\ L2*s1 - L5(c4(c2*s1*s3 + c3*s1*s2) + s4(c2*c3*s1 - s1*s2*s3)) + L3*c2*s1 \\ L1 + L3*s2 + L5(c4*(c2*c3 - s2*s3) - s4*(c2*s3 + c3*s2)) \\ 1 \end{bmatrix}$$

5.3. Cinemática inversa.

5.3.1 Introducción.

Para un control posicional del robot se deben obtener las variables articulares (ángulos de las articulaciones) para una posición y orientación espacial deseada, generalmente expresada en el espacio cartesiano. Este cálculo se denomina la cinemática inversa. En general, Se basa en la inversión de las ecuaciones obtenidas por la cinemática directa.

Debido a las no linealidades y al mayor número de grados de libertad articulares habrá casos en los que exista más de una solución y habrá que elegir la deseada. Se aprovechará la morfología del robot para simplificar la resolución del problema.

Una vez obtenidas las variables articulares se calcularán los ángulos de los servos (el programa convertirá esos ángulos a una señal PWM).

5.3.2 Conceptos y definiciones.

En este apartado se describen los conceptos relativos a la cinemática del brazo robótico utilizados en la descripción y los cálculos. También se describen otros conceptos interesantes en la robótica que se tienen en cuenta en apartados posteriores.

Articulación: unión entre dos piezas del robot (eslabones) con una libertad de movimiento asociada a cada una de ellas. En este caso cada articulación lleva asociado un grado de libertad e irá unida a un actuador.

Eslabón: elemento rígido del robot situado entre dos puntos articulares (un punto articular si es el eslabón final).

Cadena cinemática: conjunto de eslabones y articulaciones que forman el robot. De la cadena cinemática se extraen los grados de libertad del robot que son una de las características fundamentales.

Espacio cartesiano: Representación del espacio donde la posición de un punto queda unívocamente definida por su distancia a un origen medida sobre los ejes XYZ. A su vez, una orientación queda definida mediante los ángulos de giro sobre los planos que contienen a dichos ejes.

Espacio articular: representación del espacio donde la posición de un punto y una orientación asociada queda definida mediante el ángulo de giro de cada una de las articulaciones. Los ángulos de giro se consideran **coordenadas articulares.** Por lo general esta representación no suele ser unívoca ya que suele haber articulaciones redundantes.

Posición: define el punto donde está situado el efector final. Puede referenciarse al espacio cartesiano o al espacio articular.

Pose: es el conjunto de la posición del efector final junto con la orientación del sistema coordenado asociado.

Espacio de trabajo: espacio constituido por el conjunto de puntos alcanzables por el efector final.

Accesibilidad: Conjunto de puntos alcanzables con cualquier orientación. Está mucho más restringida que el volumen de trabajo.

Accesibilidad total: Es el mismo concepto que el de accesibilidad, pero descartando aquellos puntos en el que no son accesibles debido a elementos externos como, por ejemplo, lugares donde se producen colisiones con su propia estructura del robot u otro elemento.

Límites articulares: Valores máximos y mínimos que pueden alcanzar los ángulos de cada articulación. Son debidos a límites en los accionamientos, la estructura del robot o limitaciones según la carga.

Efector final (p): se define como el punto de interés de la cadena cinemática. Lleva asociado un sistema coordenado. En el caso del brazo robótico el efector final se sitúa en el centro de la pinza y a 19cm del centro de la muñeca.

Centro de la muñeca (w): punto donde intersecan los ejes de giro de la muñeca. Su posición no depende del ángulo de giro de estos ejes por lo que será útil para resolver el problema de desacoplo cinemático.

Estructura de brazo: Es la cadena cinemática formada por los tres primeros eslabones (tronco, hombro y codo) con sus articulaciones.

Muñeca: acoplada al último eslabón de la estructura brazo. Es donde se sitúa el efector final y sobre la que se define la orientación de su sistema coordenado. Todas sus articulaciones son de giro.

Precisión: diferencia entre la posición ordenada y la realmente alcanzada por el efector final. Se puede hablar de **precisión de posición (AP)** o **precisión de orientación**. Orígenes de una mala precisión pueden ser desgastes, elasticidades, deformaciones y errores en el control.

Repetibilidad: capacidad para regresar a la misma posición a la que fue dirigido en las mismas condiciones. Es una característica fundamental en tareas que requieren la repetición continuada de operaciones. Orígenes de una baja repetibilidad pueden ser problemas mecánicos en las transmisiones, huelgos, rozamientos, histéresis.

5.3.3 Desacoplo cinemático.

Para un robot con muñeca se puede observar que al situar el centro de la muñeca (w) con su sistema coordenado asociado, su orientación puede ajustarse únicamente con las variables articulares de la muñeca. La posición de w solo depende de las variables articularse de la estructura brazo (tronco, hombro y codo). Con esto se puede decir que hay un **desacoplo cinemático en el centro de la muñeca.** Es posible separar el problema de situar el centro de la muñeca con el de su orientación.

Sin embargo, no ocurre lo mismo con el efector final (p). La posición del punto p queda definida tanto por el punto w como por la orientación del sistema coordenado. Esto implica que a la hora de calcular las variables articulares dada una posición y orientación del efector final no sea posible separar el problema en calcular las variables del brazo para esa posición y variables de la muñeca para la orientación. Existe el problema del **acoplamiento cinemático para el efector final**.

Desacoplo cinemático (Pieper, 1968), cálculo del centro de la muñeca.

El objetivo es dividir el problema en el problema de la posición (estructura brazo) y el problema de la orientación (muñeca), desacoplándolos.

De la orientación se pueden obtener el vector **[n s a]** que es la matriz de transformación del sistema coordenado situado en el efector final (que orienta la herramienta) respecto al origen.

Se deduce que:

$$p = w + L5 * 0R5 \rightarrow w = p + L5 * 0R5 * z5 \qquad (1)$$

Siendo L5 la longitud de la herramienta, la pinza en este caso.

Se desconoce 0R5, pero se puede observar que solo nos interesa la tercera columna ya que se multiplica por z5 = $[0\ 0\ 1]^T$.

Obtención de 0R5 a partir de [n s a].

$$[n\ s\ a] = 0R6 = 0R5 * 5R6 = 0R5 * R_{z5,\theta 6} = 0R5 * \begin{bmatrix} \cos(\theta 6) & -sen(\theta 6) & 0 \\ sen(\theta 6) & \cos(\theta 6) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Para cumplir esta igualdad las terceras columnas de ${}_0R^5$ y ${}_0R^6$ deben ser iguales. Para obtener estas columnas se multiplica por z5 y z6 respectivamente.

$$0R5 * z5 = 0R6 * z6 \qquad (2)$$

Sustituyendo (2) en (1) se obtiene:

$$w = p + L5 * 0R5 * z5 = p + L5 * 0R6 * z6 = p + L5 * [n\ s\ a] * z6$$

$$w = p + L5 * a \qquad (3)$$

Ya se ha obtenido el punto de desacoplo cinemático w, este punto será utilizado para obtener los ángulos de giro del tronco, hombro y codo.

Obtención del vector a.

Para la resolución del problema de desacoplo se necesita el vector a. Este vector se obtiene a partir de los ángulos de Euler (ϕ, θ y ψ) con la secuencia ZYZ.

$$a = [\cos(\varphi)\ sen(\theta) \quad \text{sen}(\varphi)\ sen(\theta) \quad \cos(\theta)]^T \qquad (4)$$

Estos ángulos de Euler se obtienen a partir de la elevación (α) y giro de la pinza (β) aportados por el usuario.

$$\begin{gathered} \varphi = \theta_1 (giro\ del\ tronco) \\ \theta = \frac{\pi}{2} - \alpha \\ \psi = \beta \end{gathered} \qquad (5)$$

Para hallar estos ángulos hace falta el ángulo de giro del tronco (θ_1) que todavía no se ha hallado porque en principio se necesita el punto w. Se puede comprobar que para este robot en particular este ángulo se puede hallar con el punto p (p y q se encuentran en el plano con el que se calcula θ_1).

Ya se ha resuelto el desacoplo cinemático y se procederá a calcular las variables articulares y los ángulos.

5.3.1 Solución de θ_1 (base o tronco).

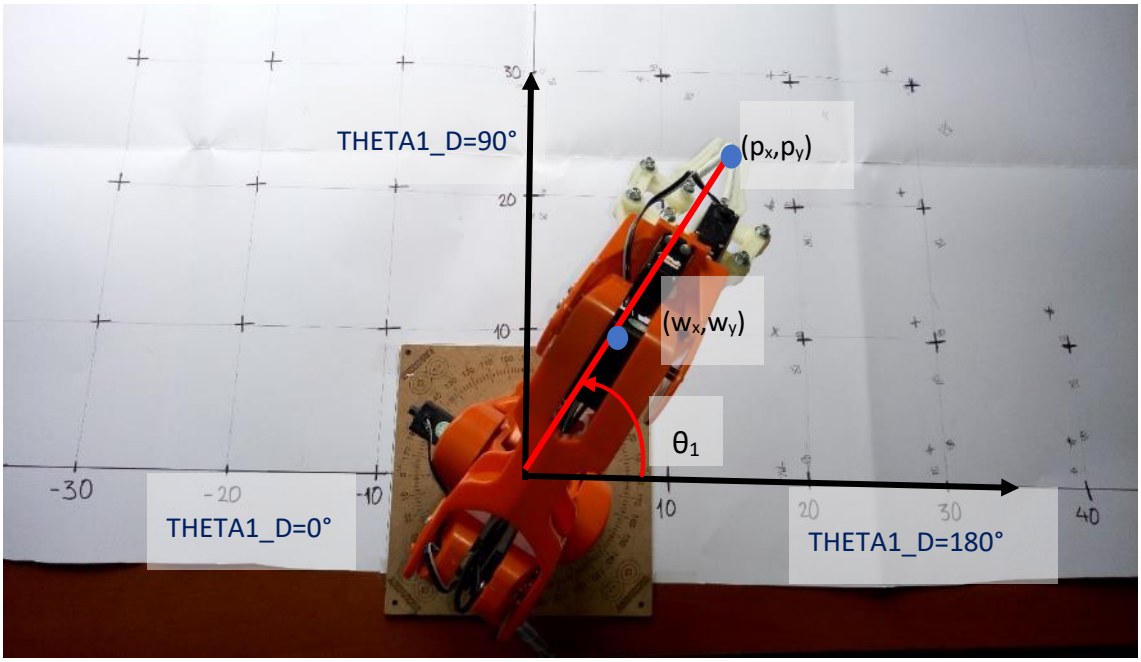


Figura 23 Cinemática inversa. Cálculo de θ_1.

Se observa que la variable articular es la arcotangente de w_y entre w_x o de p_y entre p_x. La función arcotangente presenta un inconveniente, solo devuelve valores entre $\pi/2$ y $-\pi/2$ ya que al introducir el cociente no se hace distinción de los signos de cada coordenada por separado. Es necesario utilizar la **función atan2** en la que se introducen las dos coordenadas por separado y devuelve el ángulo situado en el cuadrante correspondiente, teniendo en cuenta los signos de las coordenadas.

$$\theta_1 = \text{atan2}(w_y, w_x) = \text{atan2}(p_y, p_x) \qquad (6)$$

El ángulo del servo base es:

$$THETA1\_D = 180° - \theta_1[°] \qquad (7)$$

Distinción brazo adelante o atrás:

Brazo adelante y brazo atrás son dos configuraciones en las que el brazo robótico puede alcanzar un punto. Esto tiene influencia en las variables articulares de la estructura brazo. Para cambiar de brazo adelante a brazo atrás habría que girar 180° la base y variar los demás ángulos hasta alcanzar el punto.

El robot Tinkerkit solo admite giros del tronco entre 0 y 180° por lo que para brazo adelante solo podría alcanzar los puntos con y>=0 y para brazo atrás solo los de y<=0. Como el plano de trabajo contiene las y positivas el comportamiento del robot será siempre de brazo adelante, utilizando la fórmula (7).

5.3.2 Solución de θ_3 (codo).

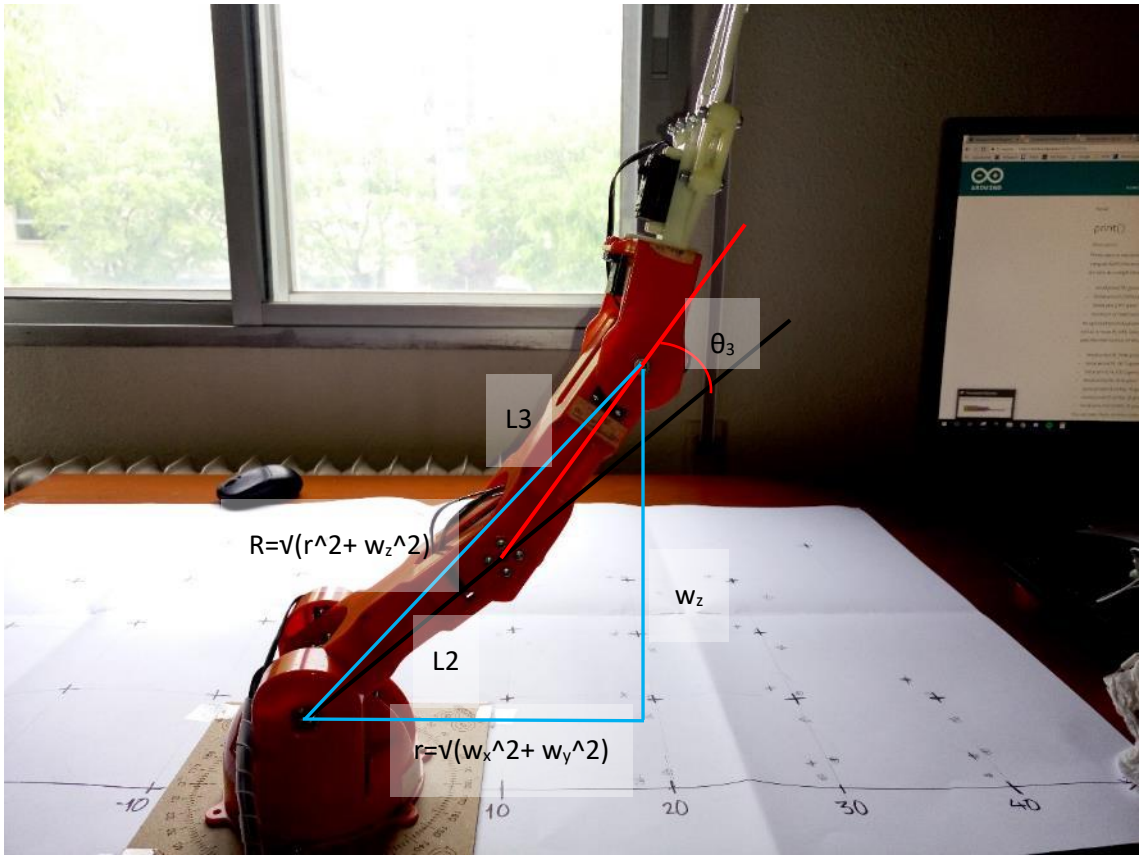


Figura 24 Cinemática inversa. Cálculo de θ_3.

Aplicando el teorema del coseno en el triángulo de lados R, L3 y L2 se despeja cos(θ_3) y se obtiene:

$$\cos(\theta_3) = \frac{{w_x}^2 + {w_y}^2 + {w_z}^2 - {L_2}^2 - {L_3}^2}{2 * L_2 * L_3} \qquad (8)$$

Es conveniente comprobar que el coseno no sea mayor que uno ya que de no cumplirse el punto no sería alcanzable de ninguna manera.

$$\text{sen}(\theta_3) = \pm\sqrt{1 - \cos^2(\theta_3)} \qquad (9)$$

Distinción codo arriba o abajo:

Codo arriba es aquella configuración en la que el brazo y antebrazo forman un ángulo tal que el codo es un vértice que apunta hacia arriba. En codo abajo el codo apuntaría hacia abajo. A diferencia de brazo arriba y abajo, el robot puede tomar ambas configuraciones y se elegirá una u otra dependiendo de la aplicación.

Al obtener el seno se observa que dos soluciones son posibles. Con la configuración de brazo adelante, la solución con seno positivo corresponde a la posición con codo abajo y la de seno negativo a codo abajo. Si se elige brazo atrás las soluciones se invierten. Se genera una variable CODO que contendrá el signo del sen(θ_3).

La solución para θ_3 es:

$$\theta_3 = CODO * \text{atan2}(\sqrt{1-\cos^2(\theta_3)}, \cos(\theta_3)) \qquad (10)$$

Ángulo del servo codo:

$$THETA3\_D = 90° - \theta_3[°] \qquad (11)$$

5.3.1 Solución de θ_2 (hombro).

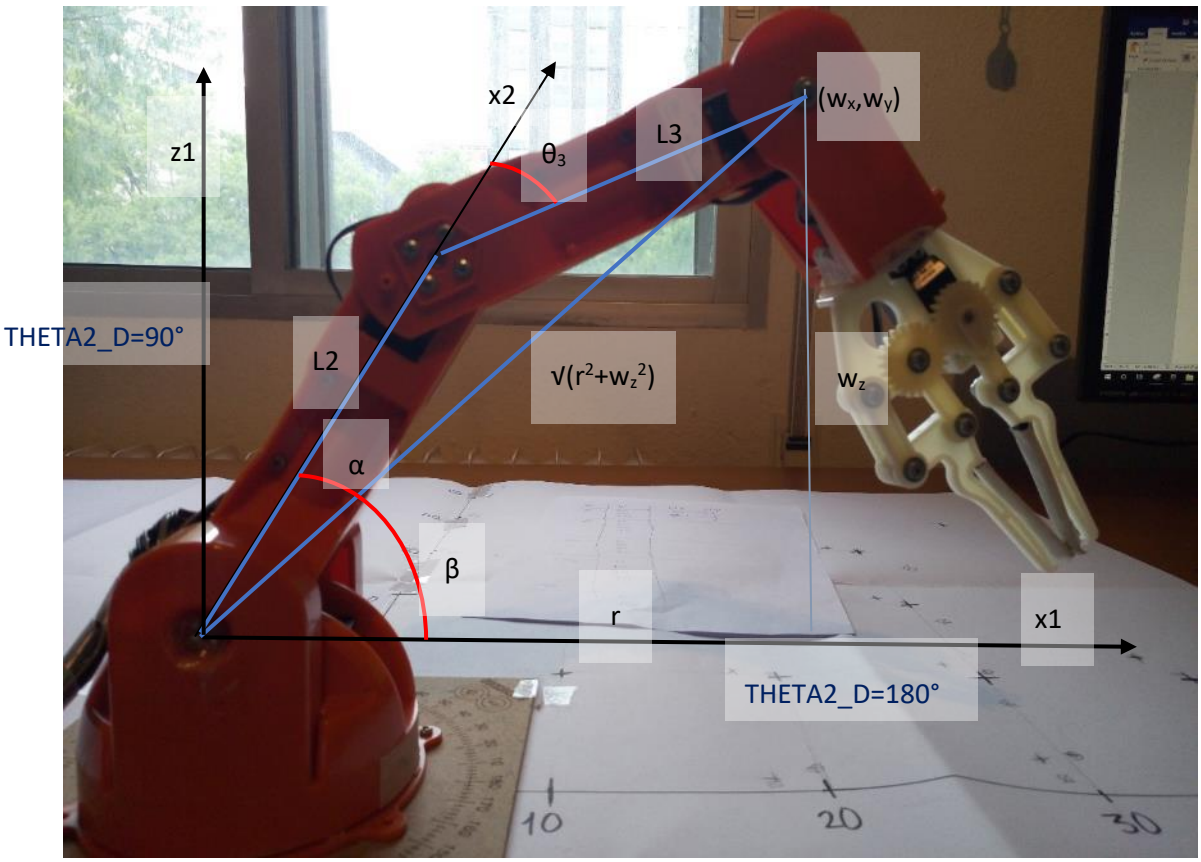


Figura 25 Cinemática inversa. Cálculo de θ_2.

Cálculo de la variable articular θ_2:

$$\theta_2 = \alpha + \beta$$

$$\theta_2 = \operatorname{atan}2\left(w_z, \sqrt{{w_x}^2 + {w_y}^2}\right) - \operatorname{atan2}(L3 * \operatorname{sen}(\theta_3), L2 + L3 * \cos(\theta_3)) \qquad (12)$$

La dependencia de la posición del codo en esta fórmula aparece en el sen(θ_3), donde para calcular el ángulo 3 ha habido que decidir entre codo arriba o codo abajo.

Observando como varía el ángulo del servo en función de θ_2 se obtiene:

$$THETA2\_D = 180 - \theta_2[°] \qquad (13)$$

5.3.2 Solución de θ_4 (ángulo de giro vertical de la pinza).

El ángulo de la muñeca vertical se puede hallar a partir de los anteriores y la elevación.

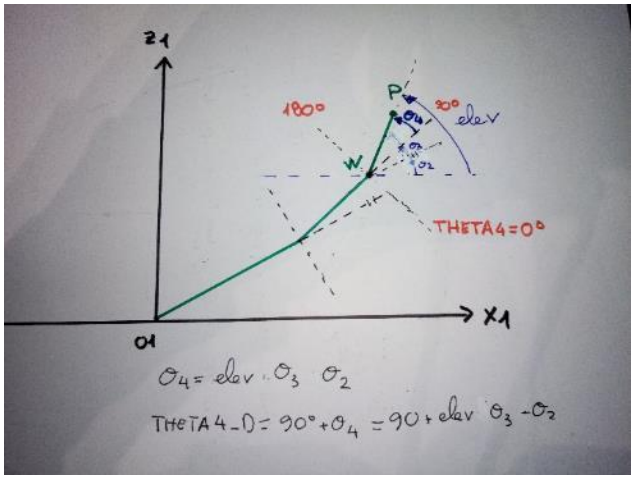


Figura 26 Cinemática inversa. Cálculo de theta4.

Ángulo del servo muñeca vertical:

$$THETA4\_D = elev - \theta_2[°] - \theta_3[°] + 90 - = elev + THETA2\_D + THETA3\_D - 180 \qquad (14)$$

5.3.3 Solución de θ_5 (ángulo de giro de la pinza).

El recorrido del servo del giro de la muñeca no es de 180º, esto provoca que la relación entre la variable articular y el ángulo es lineal, pero de pendiente no unitaria.

Cuando la pinza está paralela al suelo el ángulo es 70º y se busca que la variable articular sea 0.

Cuando la pinza está perpendicular el ángulo es 180º y se busca que el ángulo sea 90 o -90. Ya que esta posición solo es alcanzada en 180º y es una posición que puede ser recurrente en los programas, se asignará la variable articular 90º, invirtiendo así el sentido de giro positivo. A efectos prácticos, por la simetría de la pinza, el resultado es el mismo.

$$THETA5\_D = a + b * \theta_5[°]$$

$$b = \frac{180 - 70}{90 - 0} = 1.222$$

Ángulo del servo muñeca giro:

$$THETA5\_D = 70 + 1.222 * \theta_5[°] \qquad (15)$$

5.3.4 Aplicación al Robot

En principio, el usuario introducirá los siguientes parámetros:

Posición del efector final (p): mediante 3 coordenadas cartesianas xyz respecto al origen.

- **Orientación de la pinza:** mediante dos coordenadas.
- **Elevación:** expresa el ángulo que forma la pinza respecto al plano xy.
- **Giro de la pinza:** expresa el ángulo formado entre x_4 y x_5.

El programa calculará las variables articulares necesarias para alcanzar dicha posición y trasladará esas esas variables a posiciones de los servomotores (otros ángulos). Tras el cálculo de cada una de ellas se comprobará si se encuentra dentro de los límites articulares. Con que una de ellas esté fuera de los límites se dejará de calcular y el punto no será accesible.

5.3.5 Problema de la elevación

A la hora de controlar el robot con el algoritmo detallado anteriormente la mayoría de los puntos proporcionados no eran alcanzables. La longitud de la pinza es muy grande y provoca que los puntos p y w sean muy distintos (recordar que θ_1 y θ_2 se hallan con el punto w), por ello pese a que el punto p es aparentemente alcanzable, hay que ajustar bien la elevación para que realmente lo sea.

Teniendo en cuenta que el objetivo es que el robot coja posicione objetos, la elevación pierde importancia en favor de la posición del efector final. Por ello se puede hacer que el programa obtenga una elevación para la cual dicho punto sea alcanzable.

Para solventar este problema se modificará el algoritmo de la siguiente manera:

1. El usuario introducirá la posición y el giro del punto deseado.
2. El programa calculará la cinemática inversa con distintas elevaciones.
3. Una vez que el punto calcula la posición accesible el robot se moverá a dicho punto.
4. Si tras las iteraciones con la elevación no se obtiene ningún punto posible esa posición no será accesible.

Formas de realizar la iteración con distintas elevaciones.

Una forma posible es:

El programa ensaye el algoritmo con elevaciones aleatorias mediante la función *random()*.

La problemática está en que se tomen por no alcanzables puntos más fácilmente ya que no se asegura que la función random() recorra todo el abanico de elevaciones posible. A su vez, los tiempos de espera están más equilibrados y no dependen tanto de la posición del punto.

Otra forma es:

Se comienza con una elevación inicial y se suman pasos de iteración hasta llegar a la elevación final. Se escoge una elevación inicial de -90º ya que es la idónea con la que empezar si se quieren coger cosas del suelo. La elevación final se toma 90º para tener el mayor volumen de trabajo posible y descartar puntos que, debido a las características constructivas del robot (limitación de giros de las articulaciones), no se pueden alcanzar.

Se escoge la segunda forma por las ventajas comentadas anteriormente.

Ventajas del algoritmo:

- Se consigue alcanzar un rango de puntos más amplio.
- Los tiempos de espera en ningún caso superan los 100ms.

Desventajas:

- Tiempo de espera largo si el punto no es alcanzable.

Para solucionar los tiempos de espera largos si el punto no es alcanzable se limitará el espacio de trabajo mediante una condición sobre las coordenadas entregadas por el usuario.

5.4. Mejora del posicionamiento.

Una vez programada la cinemática inversa el robot ya se podía controlar mediante coordenadas xyz. Sin embargo, a la hora de mover el robot a las diferentes posiciones se observó que había una pérdida de precisión. Las causas eran:

- Huelgos y rozamientos.
- Deformaciones de los eslabones.
- Pérdida de precisión debido al redondeo en los grados o que las librerías no estaban bien diseñadas.

La pérdida de precisión impedía el agarre de objetos previamente colocados ya que la imprecisión podía llegar a un par de centímetros. Se necesitaba poder corregir este error para dotar al brazo robótico de mayor funcionalidad, con un posicionamiento lo más preciso posible.

Se pensó la forma de prever el error y corregirlo a priori. Esto requería que el error fuera constante, es decir una alta repetibilidad. La repetibilidad mayor se da si el punto de partida es el mismo, al definir una posición de partida como la posición casa se consigue esta cualidad.

Ahora que se tiene la repetibilidad hace falta el algoritmo que corregirá el punto, variando las coordenadas xyz y hacer la cinemática inversa con estas nuevas coordenadas.

5.4.1 Ensayo de precisión y medición del error.

Se crea un plano en el que se indican los ejes y se marcan puntos de interés a los que se moverá él robot. Se coloca el robot en el plano de manera que su origen coincida con el del plano.

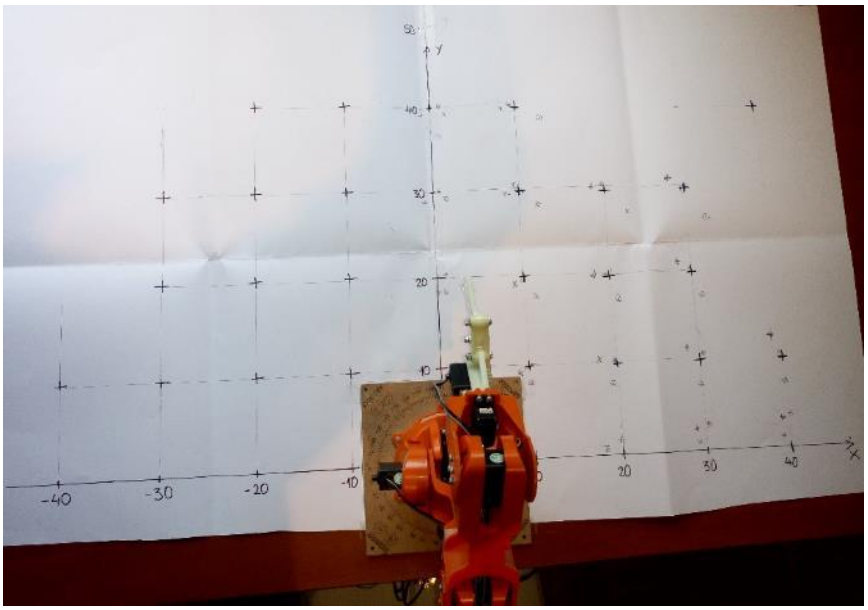


Figura 27 Plano con robot colocado.

Se envía a las posiciones marcadas en el plano y se marcan las posiciones que realmente alcanza. Se calcula el error (con sus componentes en x,y) midiendo la distancia desde el punto deseado al punto alcanzado. Siempre partiendo de la posición casa.

Forma de medir el error.

- **Posición deseada:** es aquella que se desea que alcance el robot medida en coordenadas absolutas xyz.
- **Posición alcanzada:** es aquella que realmente alcanza el robot medida en coordenadas absolutas xyz.
- **Error de posición**: Es la resta de las coordenadas de la posición deseada menos la posición alcanzada. Es un vector de 3 elementos error_x, error_y, error_z.

$$p_{deseada} = p_{alcanzada} - error \rightarrow \begin{bmatrix} error_x \\ error_y \\ error_z \end{bmatrix} = \begin{bmatrix} p_{a_x} \\ p_{a_y} \\ p_{a_z} \end{bmatrix} - \begin{bmatrix} p_{d_x} \\ p_{d_y} \\ p_{d_z} \end{bmatrix} \qquad (16)$$

El error medido es el error absoluto. Se ha optado por éste en vez del error relativo ya que no aporta información para esta aplicación. Cuando se requiere alcanzar un punto, la importancia de la desviación solo depende de la distancia al punto deseado en cuestión, es decir, las exigencias sobre la precisión son comunes a todos los puntos.

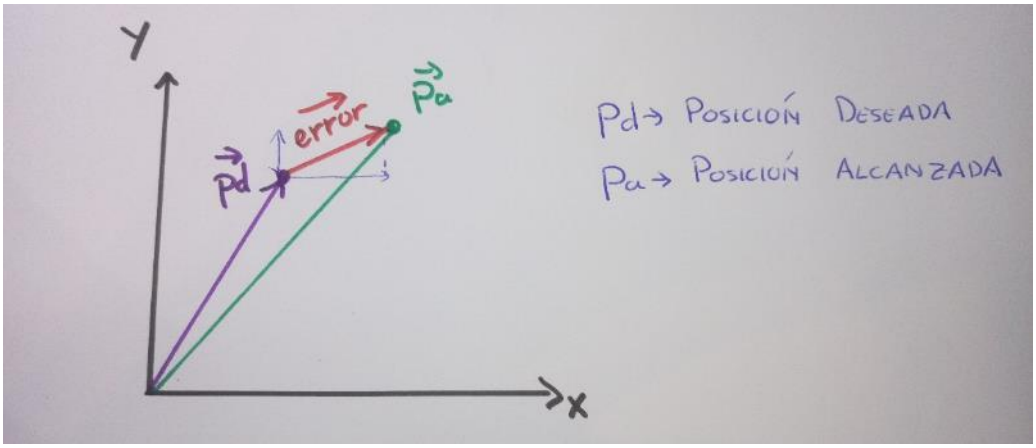


Figura 28 Forma de medir el error.

Realización del ensayo

El ensayo se realizará en el plano z=0 o suelo, es decir, no se va a ver la influencia del cambio de altura para el error posicional. Se tomarán los errores posicionales definidos en el apartado anterior, en la fórmula (16).

Se realizan dos ensayos diferentes, uno para coordenadas con x positivas y otro para x negativas. La razón de esto es que partiendo de esa posición casa el giro de la base se ejecuta en un sentido distinto en cada caso.

En los ensayos a la hora de alcanzar un punto, **siempre se partirá de la posición casa**.

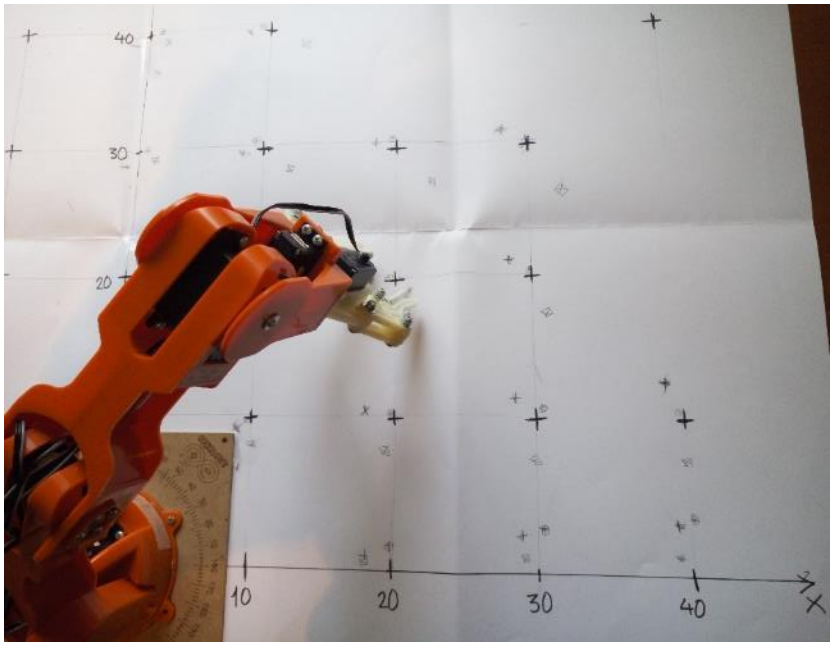


Figura 29 Robot alcanzando un punto en el plano.

5.4.2 Error para semiplano derecho (x>=0).

Tras la realización del ensayo y la anotación de los errores de la forma descrita en la Figura 28 se van a analizar los datos obtenidos, sus medias y desviaciones típicas para ver cuál es la mejor manera de aproximar mediante una función el error.

En las tablas con los puntos se observa que la desviación típica del error para x=cte. (columnas derechas) es mayor que del error para y=cte. (filas inferiores). Esto se traduce en que para x=cte. habrá mayor distancia entre los errores y el valor medio (con el que se pretende corregir). Se usarán las medias en función de x, para y=cte. para sacar la función de corrección.

Tras el ensayo se han recogido estos datos sobre los errores.

Error x

Máximo (cm)	0
Mínimo (cm)	-2
Media (cm)	-1.3
Desviación típica (cm)	3.6647

Tabla 8 Datos estadísticos sobre el error en x cometido para el semiplano derecho.

Se calcula la media de los errores para cada valor de x y se representan en la gráfica. La justificación de esto es que la desviación típica del error para x=cte. es mayor que del error para y=cte., lo que implica que los puntos con la misma x son menos dispersos y la media se aproxima mejor todos ellos. Esto se observa en las tablas del anexo.

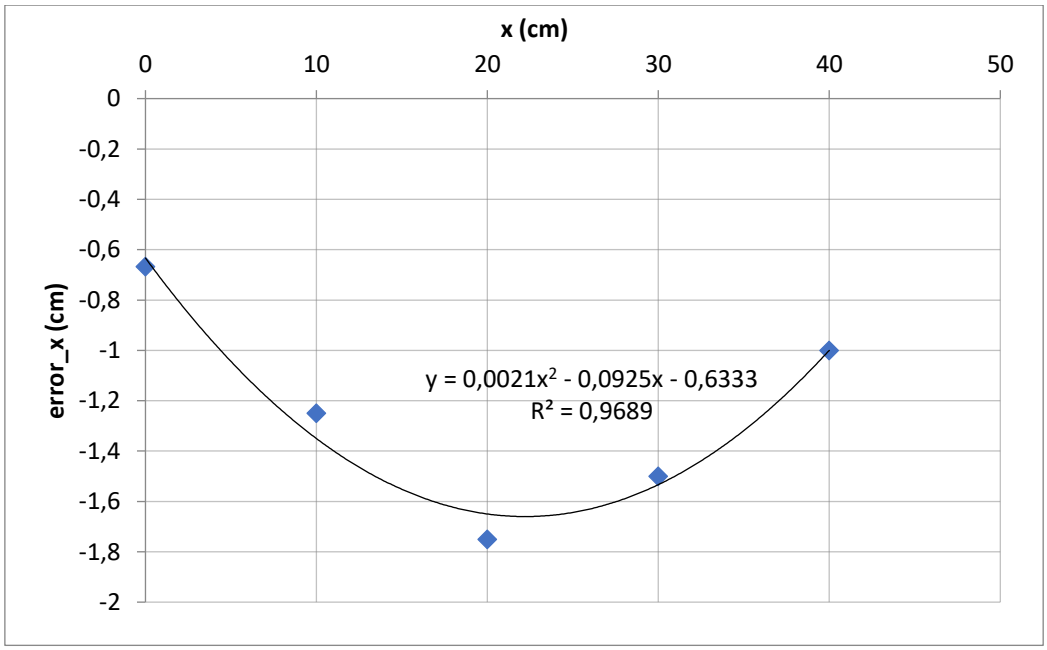


Figura 30 Medición del error en semiplano derecho. Error_x (medio) con respecto a x.

Esta fórmula del error se calculará en función de la coordenada X del punto demandado, calculando el error en X. Un ajuste polinómico de grado 2 es la mejor opción ya que se aproxima bien a los puntos.

$$error_x(x) = 0.0021 * x^2 - 0.0925 * x - 0.6333\ cm \qquad (17)$$

Error y

Máximo (cm)	3.3
Mínimo (cm)	-1.5
Media (cm)	0.36
Desviación típica (cm)	31

Tabla 9 Datos estadísticos sobre el error en y cometido para el semiplano derecho.

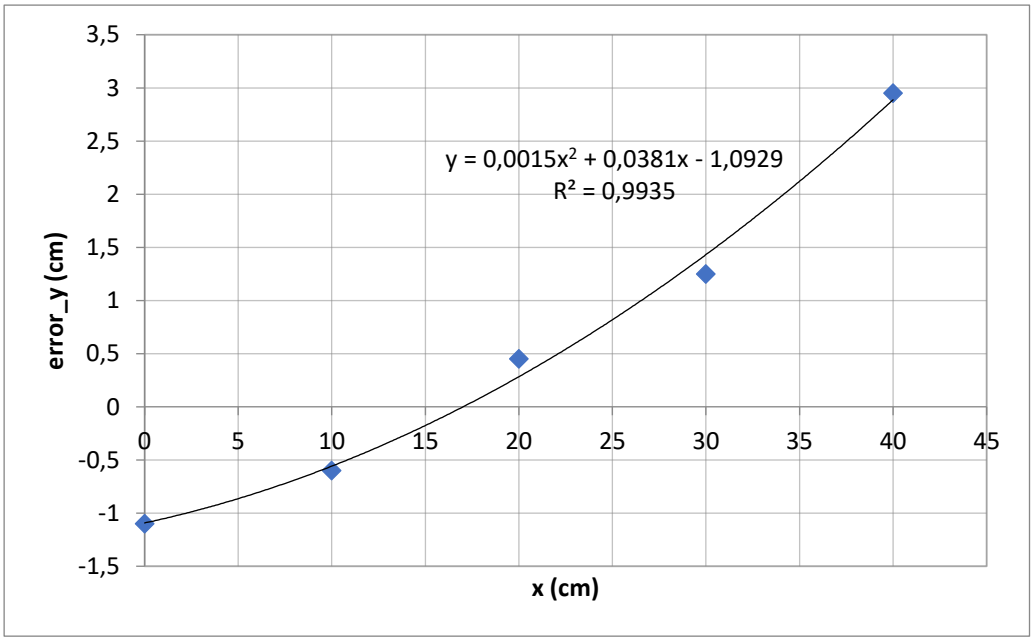


Figura 31 Medición del error en semiplano derecho. Error y (medio) respecto a x.

Fórmula del error:

$$error_y(x) = 0.0015 * x^2 + 0.0381 * x - 1.0929 \qquad (18)$$

5.4.3 Error para semiplano izquierdo (x<0).

Error x

Máximo (cm)	4
Mínimo (cm)	-1.2
Media (cm)	1.9444
Desviación típica (cm)	35.5988

Tabla 10 Datos estadísticos sobre el error en x cometido para el semiplano izquierdo.

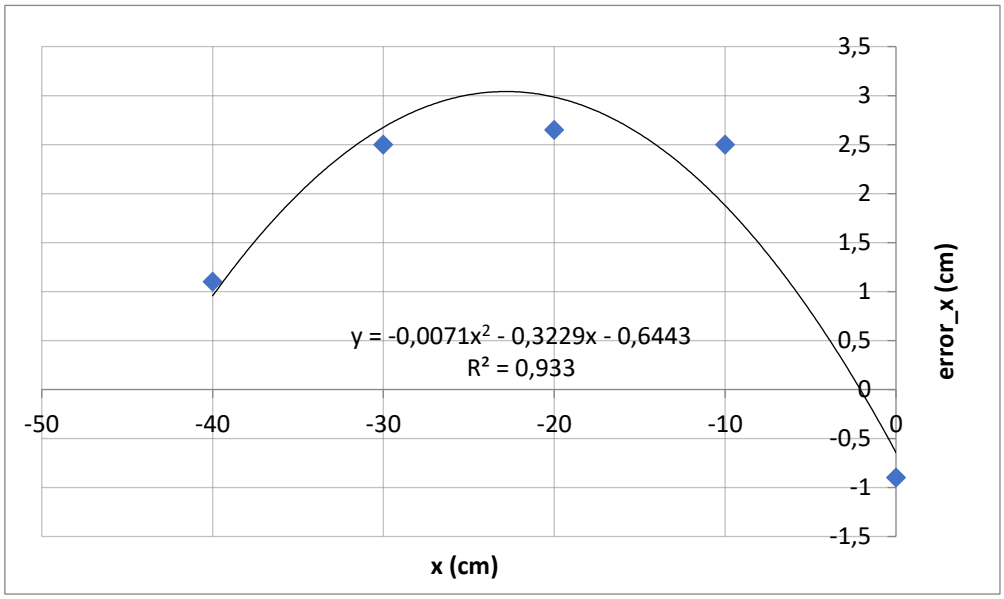


Figura 32 Medición del error en semiplano izquierdo. Error x (medio) respecto x.

Error y

Máximo (cm)	2.8
Mínimo (cm)	-2
Media (cm)	0.7000
Desviación típica (cm)	30.4000

Tabla 11 Datos estadísticos sobre el error en y cometido para semiplano izquierdo

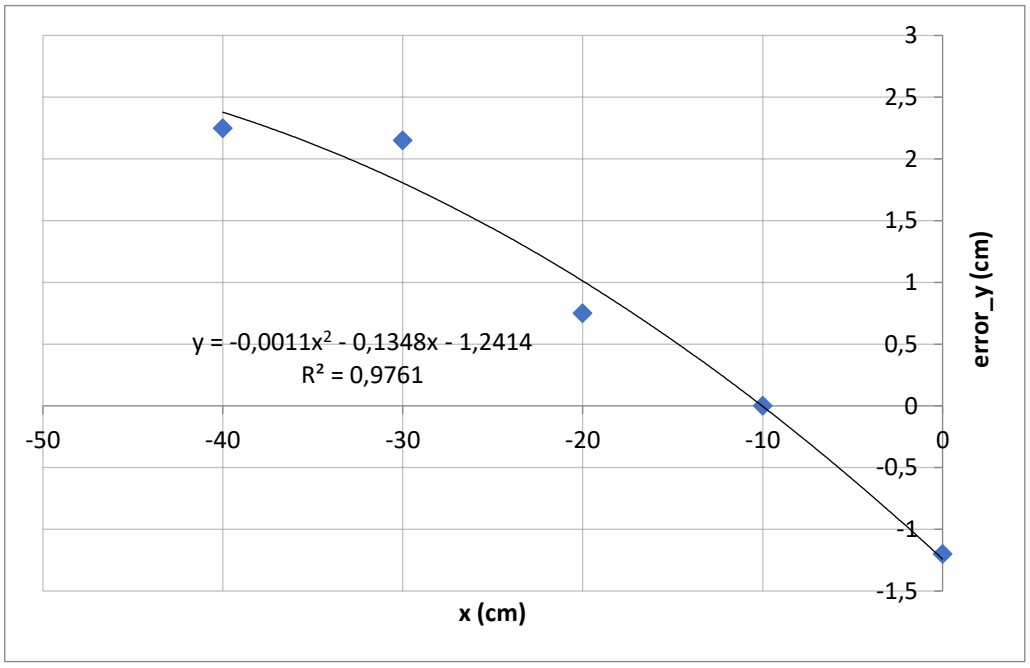


Figura 33 Medición del error en semiplano izquierdo. Error y (medio) respecto x.

5.4.4 Algoritmo de corrección.

Obtención del error

Utilizando las fórmulas de error halladas con las fórmulas (17) y (18) podemos calcular qué error se cometerá cuando se dé la orden de ir a un punto.

Corrección

La corrección consiste en restar el error calculado al punto deseado, dando como resultado un nuevo punto deseado. Se puede comprobar que ahora el punto alcanzado se aproximará mejor al punto deseado que antes.

Para que esta aproximación sea efectiva los errores del punto deseado nuevo y del antiguo deben ser similares. Como la distancia entre ellos es pequeña la variación de errores también lo será.

$$\begin{pmatrix} p_{x1} \\ p_{y1} \\ p_{z1} \end{pmatrix} = \begin{pmatrix} p_{x0} - err_x \\ p_{y0} - err_y \\ p_{z0} - err_z \end{pmatrix} \qquad (19)$$

$$\begin{pmatrix} p_a \\ p_a \\ p_a \end{pmatrix} = \begin{pmatrix} p_{x1} + err_x{}' \\ p_{y1} + err_y{}' \\ p_{z1} + err_z{}' \end{pmatrix} = \begin{pmatrix} p_{x0} + err_x{}' - err_x \\ p_{y0} + err_y{}' - err_y \\ p_{z0} + err_z{}' - err_z \end{pmatrix} = \begin{pmatrix} p_{x0} + \Delta err_x \\ p_{y0} + \Delta err_y \\ p_{z0} + \Delta err_z \end{pmatrix} \qquad (20)$$

5.4.5 Perdida del área de trabajo en la corrección del error.

La primera observación que se hizo fue que puntos cercanos al eje x (p_y cercana a 0) dejan de ser alcanzables. La explicación está en que al aplicar la corrección los puntos finales tenían coordenadas y<0, exigen una rotación de la base mayor que sus límites.

Para solucionar este problema no se aplicará la corrección a puntos cuya coordenada y sea menor que un valor, habrá que obtener el valor óptimo.

Para x>=0

Hay que buscar: el valor p_{y1} tal que: $p_{y1} = p_{y0} - err_y(x) \geq 0cm \ \ \forall x \in [0,40]$

En la Figura 31 se observa que la función $err_y(x)$ es monótona creciente en este intervalo. Su máximo se da en el extremo x=40 y es de valor $err_y max = err_y(x = 40) = 2.83\ cm$. Se toma el valor p_{y0}=3 que cumple la condición.

Para x<0

De la Figura 33: $err_y max = err_y(x = -40) = 2.4\ cm$

No se aplicará la corrección del error a los puntos cuya coordenada y sea menor de 2.6 (para dar cierto margen).

5.4.6 Problema con puntos cercanos al eje x.

Hay puntos cerca del eje y que no son alcanzables ya que el robot tiene un huelgo bastante grande en el giro de la base y no consigue llegar al límite articular. Así para aumentar la precisión en estos puntos se mandará el robot al mismo punto, pero con coordenada y=0, forzándolo siempre a llegar al límite articular.

Condición: $si\ 0cm \leq p_{y0} \leq 1cm\ ,\ entonces\ p_{y1} = 0cm$ (Se cumple tanto para x positivas y negativas)

5.4.7 Aplicación del algoritmo.

Ahora se aplicará el algoritmo y se volverán a tomar los errores de la misma manera para ver si la corrección ha sido efectiva.

Error x

Máximo (cm)	1
Mínimo (cm)	-0.6
Media (cm)	-0.0111
Desviación típica (cm)	3.8424

Tabla 12 Datos estadísticos sobre el error en x tras la corrección.

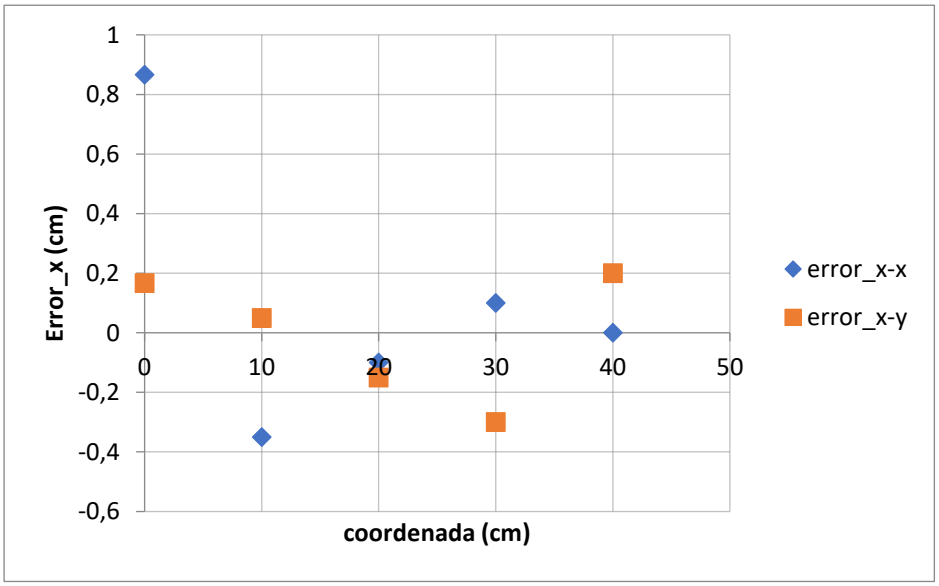


Figura 34 Aplicación del algoritmo. Error_x en función de x e y.

Error y

Máximo (cm)	3.6
Mínimo (cm)	-0.5
Media (cm)	0.3444
Desviación típica (cm)	18.2812

Tabla 13 Datos estadísticos sobre el error en y tras la corrección.

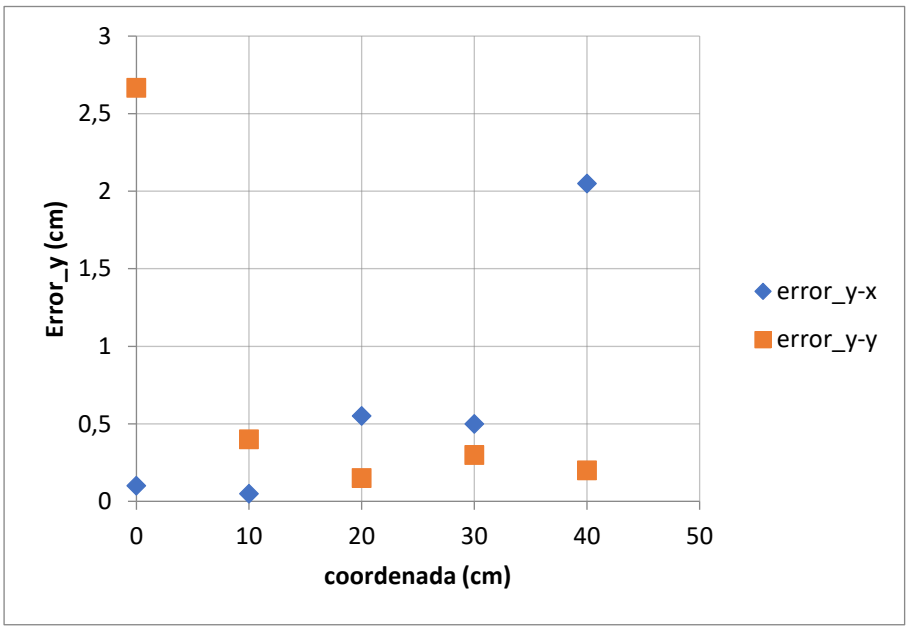


Figura 35 Aplicación del algoritmo. Error_y en función de x e y.

De la lectura de los datos recogidos en los gráficos y tablas anteriores se pueden hacer estas observaciones.

- Tras la aplicación de la corrección del error se observa que el error medio ha disminuido a 0.01 para el error en x y 0.34 para el error en y. La corrección del error ha sido efectiva.

- El error y para x=40 (solo son alcanzables puntos con y pequeña) se ha reducido de 3 a un valor de 2, que sigue siendo un valor alto. Esto ya no es subsanable por lo comentado sobre la perdida de accesibilidad.

- En el punto cercanos al eje y (x=0) el error no ha disminuido. Esto se debe a que el servo de la base no realiza ningún movimiento para alcanzar estas posiciones cuando parte de la posición casa. Se puede corregir este problema colocando una condición para que **no aplique corrección del error si se encuentra cercano al eje x, la condición es x<1.1cm.**

- El error en z se ha ido variando a lo largo del ensayo, había que elevar la coordenada z del punto deseado 2cm para alcanzarlo. El resultado es **err_z=-2cm** para todos los puntos.

5.4.8 Algoritmo final.

En base a lo explicado en cálculos anteriores, teniendo en cuenta el cálculo de errores con el problema de la pérdida de accesibilidad, se obtiene el siguiente algoritmo que queda listo para adaptarlo al lenguaje de programación de Arduino.

Si x>=0 cm

$$err_x(x) = \begin{cases} 0.0021 * x^2 - 0.0925 * x - 0.6333 & si\ x \geq 1.1 \\ 0 \quad si\ x < 1.1 \end{cases} \quad [cm] \qquad (21)$$

$$err_y(x) = \begin{cases} 0.0015 * x^2 + 0.0381 * x - 1.0929 & si\ y \geq 3 \\ 0 \quad si\ y < 3 \end{cases} \quad [cm] \qquad (22)$$

Si x<0

$$err_x(x) = \begin{cases} -0.0071 * x^2 - 0.3229 * x - 0.6443 & si\ x \leq 1.1 \\ 0 \quad si\ x < 1.1 \end{cases} \quad [cm] \qquad (23)$$

$$err_y(x) = \begin{cases} -0.0011 * x^2 - 0.1348 * x - 1.2414 & si\ y \geq 2.6 \\ 0 \quad si\ y < 2.6 \end{cases} \quad [cm] \qquad (24)$$

Para cualquier punto:

$err_z = -2$ cm

$$Si\ p_{y0} > 1, \qquad entonces \begin{pmatrix} p_{x1} \\ p_{y1} \\ p_{z1} \end{pmatrix} = \begin{pmatrix} p_{x0} - err_x \\ p_{y0} - err_y \\ p_{z0} - err_z \end{pmatrix} \qquad (25)$$

$$Si\ 0 \geq p_{y0} \leq 1, \qquad entonces \begin{pmatrix} p_{x1} \\ p_{y1} \\ p_{z1} \end{pmatrix} = \begin{pmatrix} p_{x0} - err_x \\ 0 \\ p_{z0} - err_z \end{pmatrix} \qquad (26)$$

5.4.9 Observaciones sobre la corrección del error.

- Este algoritmo se ha calculado para un brazo robótico en concreto con sus propias deformaciones, huelgos, holguras, etc. Es muy probable que este algoritmo no sea tan efectivo en otros robots.
- Este algoritmo funciona solo si el robot va al punto partiendo de la posición casa.
- El aumento de precisión es mayor para puntos más alejados del origen, sobre todo en la coordenada y.
- La implementación en Arduino consiste en añadir varios condicionales y varias operaciones con floats. El tiempo que se pierde en estas operaciones es de aproximadamente 40 microsegundos, despreciable en esta aplicación.

5.5. Agarre de objetos con la pinza

5.5.1 Introducción.

En este apartado se busca el poder agarrar diferentes objetos con la pinza. Al no contar con otros sensores, la única forma posible de agarrar objetos es que el usuario introduzca la anchura o el diámetro del objeto a agarrar y para esto se necesita una relación entre esta medida y el ángulo del servomotor necesario.

5.5.2 Ensayo y resultados.

Se ha procedido al agarre de objetos reales modificando la posición del servo (ángulo de la pinza) y tomando el que mejor se adapta. Únicamente se puede actuar sobre la posición y entonces hay un compromiso entre fuerza y posición. Si se cierra mucho la posición el servo sufre ya que no puede llegar a su posición estable, si queda muy abierto directamente no agarra el objeto.

Se busca una relación entre el diámetro del objeto y el ángulo que se debe aplicar al servo. Para obtenerla se ensayará el agarre de objetos para diferente diámetro y diferenciando entre objetos planos y objetos cilíndricos. Los resultados se representan la g

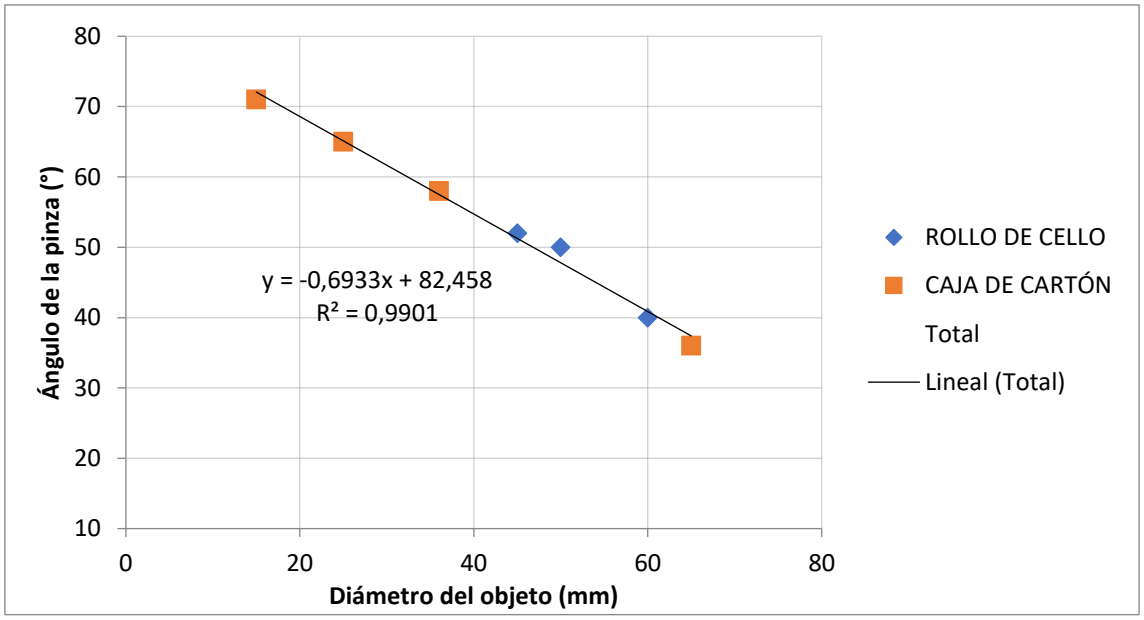


Figura 36 Gráfico ángulo de la pinza frente diámetro del objeto agarrado.

El ajuste lineal es bueno ya que la R^2 es cercana a 1. La relación lineal queda:

$$ángulo[°] = -0.6933 * diámetro[mm] + 82.458 \qquad (27)$$

Ya se tiene una relación entre el diámetro del objeto a agarrar y el ángulo del gripper, la aproximación lineal es buena. Esta fórmula se utilizará en la programación de Arduino y habrá que hacer la aproximación a entero.

5.6. Programación en Arduino.

5.6.1 Introducción.

En apartados anteriores se ha logrado obtener la manera de mover el robot con precisión a diferentes puntos del plano. Ahora solo queda traducir estos resultados al lenguaje de programación de Arduino para poder llevar a cabo los movimientos. A parte, se dotará al robot de otras funcionalidades extra, todas basadas en su posicionamiento y movimiento.

El desarrollo de la programación ha constado de tres partes diferenciadas que se describen en orden en el que se crearon.

En la primera parte, se ha dedicado exclusivamente a la programación de la cinemática inversa. Se programó una función que realizaba el cálculo de las variables articulares a partir de coordenadas y movía el robot a dicho punto. Esta parte era básicamente traducir a lenguaje de Arduino los cálculos de cinemática inversa.

En la segunda parte, una vez que ya funcionaba la cinemática inversa y ya se tenía conocimiento pleno del funcionamiento del robot, ya se crearon las demás funciones. Funciones que permitían mover el brazo robótico a coordenadas articulares, apertura y cierre de la pinza, almacenar puntos y movimiento automático entre otras.

Finalmente, solo quedó programar el programa principal para hacer efectiva la comunicación con el módulo bluetooth y el joystick. Este programa utilizaría las subrutinas programadas en las partes anteriores y no se describe en este apartado, está descrito en el apartado Comunicación Arduino con dispositivo Bluetooth.

5.6.2 Librerías.

A continuación, se nombrarán y se dará una pequeña descripción las librerías usadas en la programación del brazo robótico basado en Arduino.

Servo.h

Permite el control con Arduino de uno o varios servomotores. Esto se logra mediante la variación del ciclo de trabajo de la señal PWM. Los servos deberán ir conectados a los pines de salida analógica de Arduino.

Se declararán los servomotores como objeto cuyo dato será el ángulo en grados de dicho servo. Se podrá modificar y leer este dato.

Braccio.h

Es una librería creada para para controlar el funcionamiento de la shield del robot Tinkerkit Braccio. Hace uso de la librería Servo.h. Se usan las siguientes funciones:

- *Braccio.begin():* Inicializa las salidas de los servomotores en los pines 11,10,9,6,5,3. Pone al robot en su posición inicial.

- *Braccio.Servomovement():* Mueve el robot a las variables articulares con el tiempo entre movimientos. Son incluidos los límites de giro de manera que el robot no superara las posiciones extremas.

Ejemplo de utilización:

```
                    //(step delay, M1, M2, M3, M4, M5, M6);
Braccio.ServoMovement(20,           0,  15, 180, 170, 0,  73);
```

Step delay: tiempo que se espera tras mover cada servomotor 1°.

M1: ángulo del servo tronco o base (entre 0° y 180°).

M2: ángulo del servo hombro (entre 15° y 165°).

M3: ángulo del servo codo (entre 0° y 180°).

M4: ángulo del servo muñeca elevación (entre 0° y 180°).

M5: ángulo del servo muñeca giro (entre 0° y 180°).

M6: ángulo del servo pinza (entre 10° y 73°).

5.6.3 Variables utilizadas.

Estas son los tipos y las variables utilizadas durante la programación. Pueden ser parámetros que se introducen a las subrutinas o que devuelven tras la subrutina.

Punto en coordenadas articulares: vector de 6 elementos. Los cinco primeros elementos son las variables articulares. El sexto elemento es un número que será 1 si ese punto es accesible y 0 si no lo es. Esto último sirve para que haya que definir primero el punto antes de lanzar las funciones.

Punto en coordenadas cartesianas: vector de 4 elementos. Los 3 primeros son las coordenadas xyz, el cuarto es el ángulo de giro de la pinza en grados.

Origen relativo: vector de 3 elementos. Contienen las coordenadas xyz del origen relativo.

BRAZO y CODO: variables que definen la posición del robot a la hora de alcanzar un punto como se explica en la cinemática inversa. Por defecto BRAZO=1 y CODO=-1. Solo se podrá actuar sobre el valor de CODO.

Corrección del error (error_fix): variable char. Si se pone a 1 se ejecutará el algoritmo de corrección del error en la cinemática inversa.

Apertura: variable de tipo float. Es el diámetro del objeto que se quiere atrapar con la pinza.

Pinza: variable tipo int. Contiene el ángulo del servo de la pinza deseado.

Velocidad: una variable int que podrá tomar números del 1 al 4. Con esta variable se ejecutará una función para definir la velocidad del robot.

Matriz para guardar posiciones: Almacena puntos articulares junto con la variable articular de apertura de la pinza por filas, tiene 7 columnas. El número de filas es libre y se define al principio de la programación.

punto, punto_total, punto_max: se utilizan para recorrer la matriz que almacena los puntos guardados.

- punto: indica la fila de la matrix_art a la que se quiere acceder.
- punto_total: indica el número a partir del cual hay que resetear la variable punto a 0. Indica cuando retornar al inicio de la matrix_art.
- punto_max: indica el máximo valor que tomará punto_total. Coincide con el número de la última fila de la matrix_art.

Automático: variable tipo booleano. Si es verdadero, se ejecuta el movimiento automático con los puntos guardados. Si es falso se moverá manualmente.

5.6.4 Funciones y diagramas de flujo de la programación en Arduino.

Cinemática inversa

Esta función es a encargada de obtener los ángulos de los servomotores necesarios para alcanzar un punto cartesiano introducido mediante un vector de 4 variables elementos (xyz y giro de la pinza). Es la aplicación de las fórmulas obtenidas en el apartado 5.3 Cinemática inversa.

Consta de una primera parte en la que calcula las coordenadas absolutas del punto (en caso de que el origen relativo no coincida con el absoluto). Se comprueban que las coordenadas absolutas pertenecen al espacio de trabajo (una condición sobre las coordenadas). Tras lo anterior, con las coordenadas absoluta, calcula en ángulo de la base (necesario para obtener el centro de la muñeca).

Con las coordenadas absolutas y si el parámetro *error_fix=true*, se aplica el algoritmo de corrección del error (apartado 5.4). Es el usuario el que decide aplicar este algoritmo.

El siguiente paso es la obtención del centro de la muñeca y los demás ángulos. Aquí se realiza una iteración con elevaciones de la pinza desde -90 a 90 grados, la razón de esta forma de proceder se encuentra en el apartado 5.3.5 Problema de la elevación.

Tras la obtención de cada ángulo aplicando las fórmulas es necesario comprobar si se encuentran dentro de los límites articulares de los servomotores. De no ser así, el punto no sería accesible y se saldría de la subrutina o se probaría con una nueva elevación (solo si el ángulo depende de la posición del centro de la muñeca).

Si el programa no encuentra ninguna configuración posible el bit de accesibilidad del punto se pone a cero la variable *accesibilidad* y las funciones encargadas de mover el robot a dicho punto no actuarán.

Función: **cininv(pr,Or,elev_min,elev_max,error_fix)**
-p: vector de 4 elementos float.
-Or: vector de 3 elementos float.
-elev_min: integer.
-elev_max: integer.

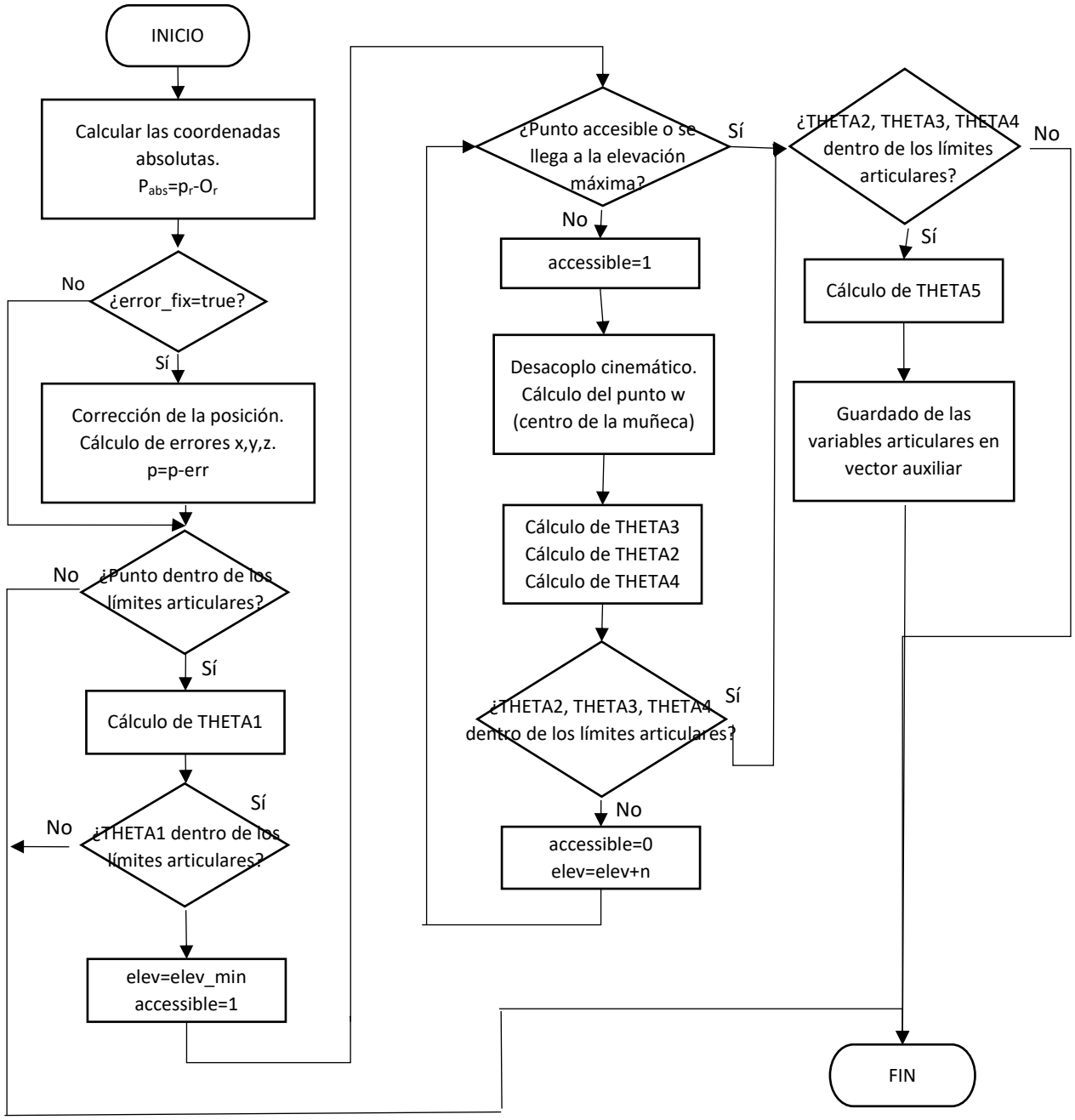


Figura 37 Diagrama de flujo de la función que calcula la cinemática inversa.

Mover el robot con variables articulares.

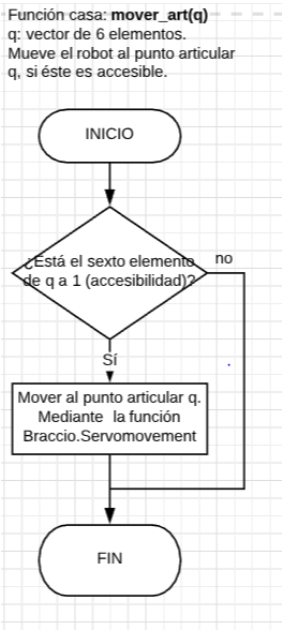


Figura 38 Diagrama de flujo de la función que mueve el robot a unos ángulos de los servomotores.

Esta subrutina consiste en mover el robot a unas coordenadas articulares (ángulos de los servomotores) que se introducen mediante un vector. Cada vector tiene un elemento que se debe poner a 1 (por el usuario o en otras funciones) cuando este punto sea accesible ya que sino no se tendría capacidad de saber si el punto ha sido calculado o simplemente proviene de inicializar la variable.

Para mover el robot a los ángulos introducidos se utiliza la función *BraccioServomovement* de la librería *Braccio.h*.

Mover a punto con coordenadas xyz y giro.

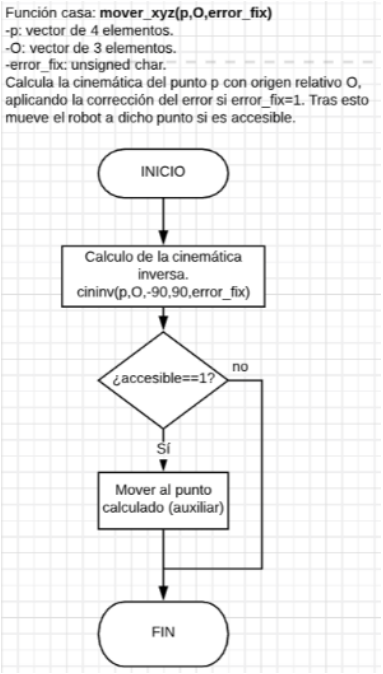
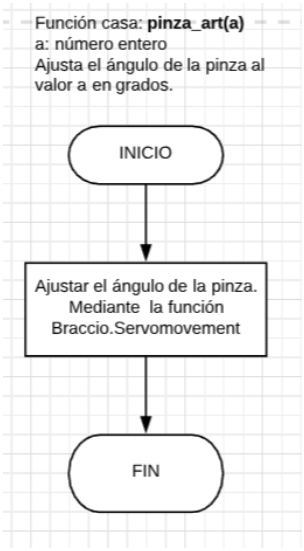


Figura 39 Diagrama de flujo de la función dadas unas coordenadas xyz, mueve el robot a dicho punto.

Con la introducción de las coordenadas del punto se calcula la cinemática inversa () y se calculan las variables articulares. Estas variables articulares se recogen en un vector de 6 elementos, se comprueba que el punto es accesible (sexto elemento a 1) y se llama a la función *BraccioServomovement* de la librería *Braccio.h* con este punto.

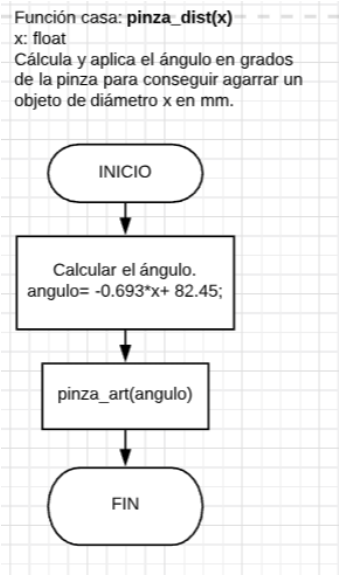
Mover la pinza con un ángulo del servo.



Esta función consiste en asignar el ángulo introducido al servomotor de la pinza. Hace uso de la función *BraccioServomovement* de la librería *Braccio.h* y no altera las demás variables articulares. Se ha decidido separar el comportamiento de la pinza y el resto del brazo robótico porque la pinza no influye en la forma de alcanzar unas coordenadas (cinemática inversa).

Figura 40 Diagrama de flujo de la función que aplica un ángulo al servo de la pinza.

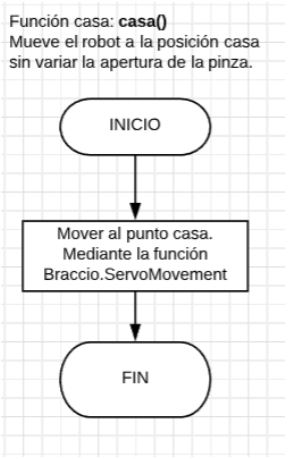
Cerrar o abrir la pinza a una distancia.



Se introduce el diámetro/anchura del objeto que se quiere agarrar con la pinza. Con esta medida se calcula el ángulo del servomotor pinza necesario (ver apartado 5.5) y se asigna este ángulo mediante la función del apartado anterior (*pinza_art*).

Figura 41 Diagrama de flujo de la función que abre la pinza a una distancia dada.

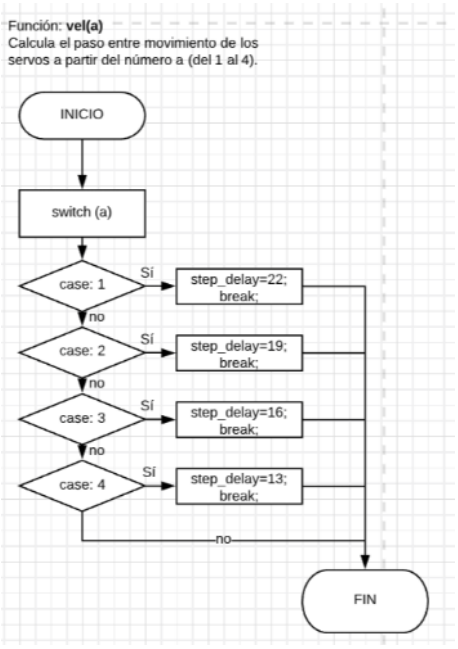
Ir a la posición casa.



Al llamar a esta función el robot se mueve a la posición casa. Se asignan a los servomotores unos ángulos predefinidos que corresponden a dicha posición. En ningún momento se altera el ángulo de la pinza.

Figura 42 Diagrama de flujo de la función que mueve el robot a la posición casa.

Variar la velocidad de movimiento.



Con esta función se logra variar la velocidad de movimiento de motor. Se introduce un número del 1 al 4 (cuanto mayor, mayor velocidad) y la variable *step_delay* su valor correspondiente. Esta variable define el tiempo en milisegundos que se espera tras variar la posición de cada servo en 1 grado siempre que se llama a la función *BraccioServomovement*.

Figura 43 Diagrama de flujo de la función de ajuste de la velocidad de movimiento.

Guardar la posición actual y ángulo de la pinza.

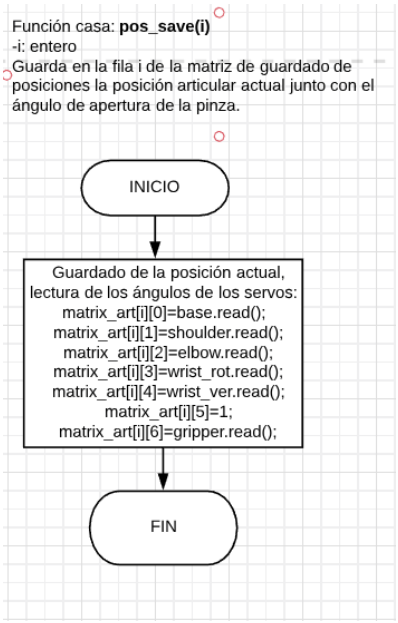


Figura 44 Diagrama de flujo de la función que almacena las variables articulares de un punto en la matriz.

Mediante esta función se guarda en la fila i de la matriz, siendo i el número entero introducido, los ángulos de todos los servomotores. Cada fila de la matriz se puede separar en un punto articular (elemento 0 al 5) y ángulo de la pinza (elemento 6). Esta matriz es la que se usa luego para el movimiento automático.

Extraer posición guardada.

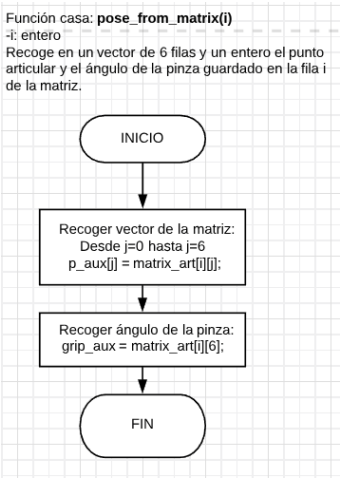


Figura 45 Diagrama de flujo de la función que recoge el punto guardado en la matriz en un vector y un entero.

Se guarda en un punto articular (vector de 6 elementos) y en una variable entero, la información guardada en la fila i-ésima de la matriz de guardado de puntos.

Este punto articular y el ángulo de la pinza podrán usarse luego para mover el robot.

Borrar posición guardada.

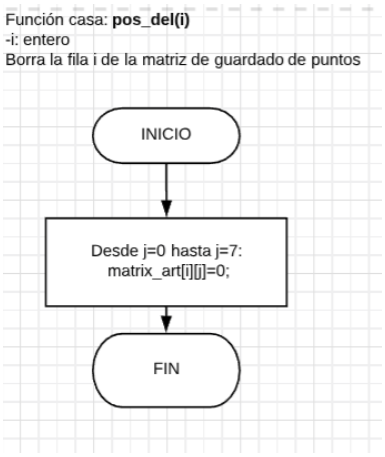


Figura 46 Diagrama de flujo de la función que borra un punto (fila) de la matriz.

Pone a 0 todos los elementos de la fila i-ésima de la matriz, es decir, borra una posición guardada.

Enviar posición actual xyz+giro.

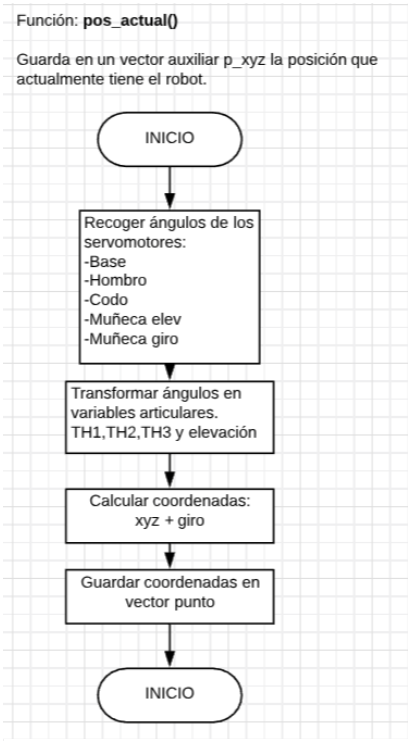


Figura 47 Diagrama de flujo de la función que calcula las coordenadas de la posición actual en un vector.

En esta posición se calcula la posición a través de los ángulos de los servomotores. Esta posición se guarda en una variable punto cartesiano (vector de 4 elementos) y, por otro lado, la elevación se almacena en una variable entero.

En el programa principal estos datos se enviarían a un dispositivo donde se mostraría la posición actual.

Movimiento automático.

Este conjunto de instrucciones va dentro del bucle *loop()* y solo se ejecuta si la variable *autom=1.* Realiza el movimiento a un punto previamente guardado en la matriz. Realiza los siguientes pasos:

1. Extrae la posición guardada en una fila de la matriz de guardado, la variable *punto* indica el número de la fila. Función: *pose_from_matrix*().
2. Se desplaza a ese punto.
3. Incrementa la variable punto para que cuando se vuelva a ejecutar se desplace al siguiente punto. Si la variable llega al máximo (*punto_total*) se reinicia a 0.

Esta rutina debe ejecutarse en intervalos de tiempo concretos, más o menos grandes, y permitiendo ejecutar las demás mientras no se ejecuta. Se captura el tiempo en milisegundo en un instante en una variable t1 mediante la función millis(). Tras cada iteración del bucle se captura el tiempo de nuevo en una nueva variable y se comprueba si la diferencia t2-t1 supera el tiempo entre movimientos (definido por el usuario). Si la diferencia es mayor se ejecuta el movimiento y se pone t1 al tiempo actual.

Utilización del joystick.

El joystick se puede utilizar de dos formas, para variar los ángulos de los servomotores o para hacer movimientos variando coordenadas xyz. Esto se realiza mediante dos rutinas que se ejecutan dentro del bucle loop(). Los pasos se describen a continuación.

<u>Joystick para movimientos articulares.</u>

1. Se comprueba la entrada conectada al botón del joystick de manera que si se pulsa se cambia de articulación.
2. Se recoge la entrada analógica de Arduino (se almacena en un entero) a la que se conecta el joystick. Si sobrepasa unos límites se continúa con el programa.
3. Se varía el ángulo de la articulación seleccionada en un grado. Esta variación será positiva o negativa dependiendo del valor de la entrada analógica.
4. Se realiza una espera calculada mediante una relación lineal con la entrada analógica. Esto dota de una velocidad variable dependiendo de la posición del joystick.

Joystick para movimientos articulares.

Lo primero es calcular el desplazamiento a realizar en cm. Este desplazamiento xyz se guardará en un vector de tres elementos *dxyz[3]*.

1. Se comprueba el botón del joystick. Si se hace una pulsación corta el desplazamiento en z es 2cm, si la pulsación es larga el desplazamiento en z es -2cm. Para distinguir entre pulsación corta y larga se utiliza la función *millis()* para obtener el tiempo entre pulsar y soltar.

2. Se recogen las entradas analógicas del joystick, una entrada se asigna al desplazamiento en x y la otra al desplazamiento en y. Si superan un valor, el desplazamiento es 1cm y si son inferiores a otro valor, el desplazamiento es -1.

3. Se calcula la posición xyz actual a partir de los ángulos de los servomotores. Se guarda en el punto *p0.*

4. Se calcula el punto de destino *p1* sumando *dxyz* a *p0*.

5. Se mueve el robot al punto *p1*.

5.7. Sistema de creación de aplicaciones MIT App Inventor.

5.7.1 Necesidad de la aplicación.

Para conectar el dispositivo móvil con el brazo robótico a través de bluetooth es necesario un interfaz que lo permita, esto se consigue a través de una aplicación. Hay opciones disponibles en la tienda de aplicaciones, pero no se adaptan bien ya que el brazo robótico cuenta con funcionalidades diversas.

Se hace necesario usar una plataforma para programar una aplicación para el móvil que permita controlar el brazo robótico. La plataforma Mit App Inventor es una buena opción debido a su sencillez y facilidad de programación.

5.7.2 Sistema MIT App Inventor.

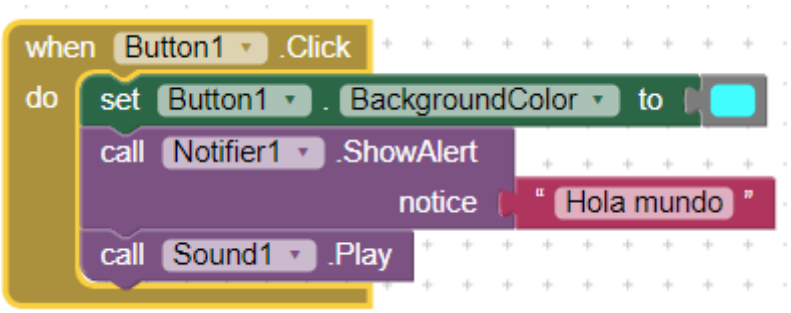
El App Inventor es un entorno de desarrollo creado por Google Labs. El sistema fue lanzado en 2012. Permite el desarrollo y distribución de aplicaciones para el sistema operativo Android de forma gratuita.

La estructura de la programación es muy visual, dejando de lado la programación por escritura de código y centrándose en la colocación de bloques. No requiere aprendizaje previo de ningún lenguaje de programación para su utilización.

En este programa por un lado está la programación por bloques y luego el diseño visual, donde se añaden y colocan los elementos en los lugares deseados.

Otra ventaja de este sistema es que permite el uso de los diferentes sensores del móvil, y funcionalidades como el bluetooth, wifi y la realización de llamadas entre otros.

Ejemplo de programación:



Cuando se pulsa el botón se cambia el color de fondo a azul, salta un mensaje de notificación y se reproduce un sonido.

Figura 48 Ejemplo de bloque de programación de MIT App Inventor.

5.8. Creación de la aplicación Bluetooth.

Se ha creado la aplicación bluetooth para Smartphone utilizando el programa MIT App Inventor 2. Se ha diseñado para comunicarse vía bluetooth con el Arduino, es decir, enviar mensajes de texto.

La aplicación en la parte superior cuenta con un botón y un texto que indica el estado de conexión y desconexión. Justo debajo está una pestaña donde se selecciona la velocidad. Más abajo cuenta con los botones de navegación y otros botones para moverse a la posición casa o a la posición anterior. Después cuenta con la parte de guardado/borrado de puntos para el movimiento automático del robot. Debajo tiene un cuadro de texto donde recibe la posición actual si se pulsa el botón.

En la parte de abajo están los controles para el movimiento del robot. Siendo posibles o control con variables articulares, control con coordenadas xyz o un selector para habilitar el control con el joystick. Cada tipo de control está en una pantalla diferente que se cambian con flechas de navegación.

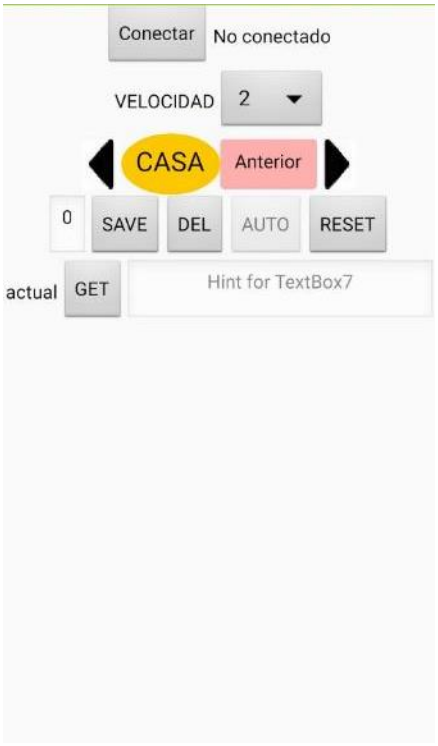


Figura 49 App para control del robot en smartphone. Pantalla cuando está desconectado.

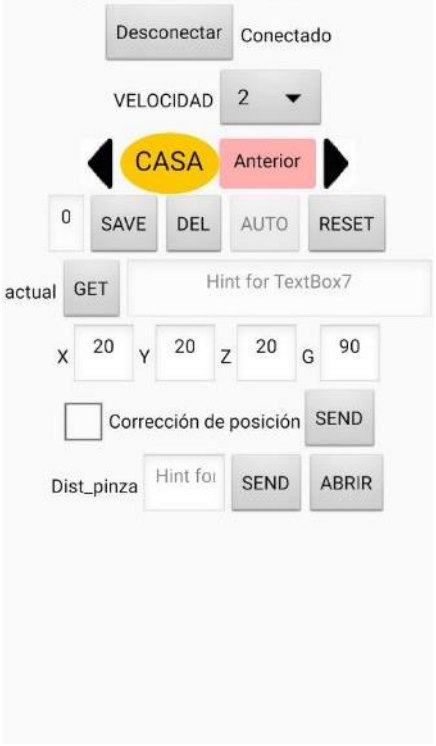


Figura 50 App para control del robot en smartphone. Pantalla cuando está conectado y en control con coordenadas xyz

Figura 51 App para control del robot en smartphone. Pantalla cuando está conectado y en control con coordenadas xyz

Figura 52 App para control del robot en smartphone. Pantalla cuando está conectado y en control con ángulos.

5.9. Comunicación Arduino con dispositivo Bluetooth.

El objetivo es a través del módulo bluetooth recibir las órdenes desde el dispositivo bluetooth y que el Arduino las ejecute. En este apartado se va a describir la forma en que estás ordenes se envían y reciben y qué rutinas ejecuta el tras la recepción.

El módulo bluetooth va conectado al puerto Serial (pines 1 y 2 de Arduino).

5.9.1 Funciones utilizadas.

Serial.print() / Serial.println(): envía por el puerto Serial el parámetro dentro del paréntesis (puede ser un carácter, un texto o un número) adaptándolo previamente a una variable texto. En el envío de variables tipo float se puede ajustar el número de decimales añadiendo una coma y el número de decimales tras la variable float. Ejemplo: Serial.print(2.136,1) devuelve 2.1)

Serial.flush(): espera a que la transmisión por el puerto Serial se complete.

Serial.parseInt(): devuelve el primer entero válido que se recibe en el puerto Serial. Caracteres iniciales no enteros o el signo menos son ignorados.

Serial.parseFloat(): devuelve el primer número en coma flotante recibido en el puerto Serial. Ignorará caracteres iniciales que nos son dígitos o el signo menos.

millis(): devuelve el número con el tiempo en milisegundos desde que se ha iniciado el programa de Arduino.

5.9.2 Mensaje que se envía desde el dispositivo Bluetooth.

El mensaje enviado desde el bluetooth se recibirá con el Arduino y el Arduino deberá actuar en consecuencia. Debido a que el programa posee varios comandos y que es posible que no se reciba algún bit se ha decidido estructurar el mensaje de esta manera.

Mensaje

Cada parte irá separada de la siguiente por el carácter de separación (‘|’).

Byte 0	Byte 1	Byte 2	Byte 3-12
START_CHAR	DIV_CHAR	COMMAND	PARÁMETROS + DIV_CHAR (opcional)

START_CHAR: Es el carácter de inicio del mensaje. El programa de Arduino leerá continuamente los caracteres recibidos hasta que este sea igual que el definido en su programa.

DIV_CHAR: Es el carácter de división. Se utiliza para poder diferenciar los números recibidos y leerlos correctamente.

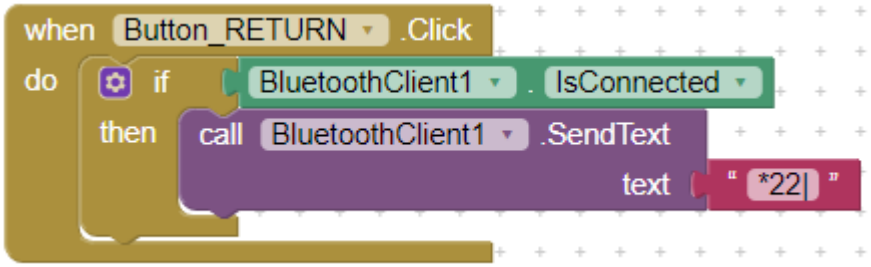
COMMAND: Es un número entero que identifica la función que ejecutará el Arduino. El programa en Arduino tiene definidos unos valores como constantes para cada comando y se comparan con el recibido.

PARÁMETROS: Son números reales o enteros que contiene la información que ha dado el usuario, como el punto al que desea llegar, la apertura de la pinza, etc.

5.9.3 Envío de mensaje desde Smartphone.

En el entorno de Mit App Inventor 2 se ha creado la aplicación. Mediante la pulsación de los diferentes botones se almacenará un string con los elementos indicados en la tabla anterior. Los parámetros se recogen de listas o sliders modificables por el usuario de la aplicación, estos números pueden requerir una adaptación o aproximación.

El envío se hace de forma automática al llamar a la función *call<bluetooth.client>Sendtext*

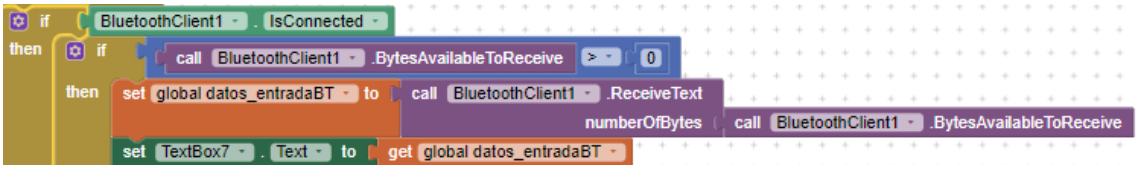


5.9.4 Recepción de mensaje en Smartphone.

Se van a explicar cómo recibir datos en el Smartphone mediante las funciones de Mitt App Inventor. Tras esto se comentarán los problemas y soluciones surgidos.

Para recibir un dato primero hay que enviar el comando a Arduino para que lo envíe, se ejecutan las mismas acciones que en el programa anterior.

Tras esto hay que recibirlo en la app. Se usa este comando:



Esto significa que, si el bluetooth está conectado, se comprueba si hay bytes disponibles para recepción. Si es verdadero, se reciben tantos bytes como estén disponibles y se guardan en una variable string. Tras esto, esta variable string puede mostrarse en un cuadro de texto.

Hay que tener en cuenta que debe pasar un tiempo entre envío del comando y recepción. Para esto se utiliza un reloj que se activa tras el envío del comando y recibe los datos cuando dispara, tras la recepción de datos se desactiva.

5.9.5 Envío desde Arduino.

Para enviar datos desde Arduino basta con almacenar el mensaje que se quiere enviar en un string y mandarlo por el puerto Serial mediante las funciones Serial.print() o Serial.println().

5.9.6 Recepción desde Arduino.

Pasos en el programa, realizados en bucle:

1. Recepción del mensaje y lectura de un carácter.
2. Comprobación del carácter de inicio. Si no coincide con el definido, se siguen leyendo caracteres hasta encontrar uno que coincida.
3. Lectura del comando (número entero) mediante Serial.parseInt() o Serial.parseFloat(). Dependiendo del comando se ejecutará una acción u otra.
4. Lectura de demás parámetros (floats o enteros).
5. Ejecutar la acción correspondiente. Por ejemplo: mover robot a punto, activar modo automático o variar velocidad.

Ya se ha creado el programa para el envío y recepción de datos por ambas partes. Ahora solo falta llamar a las funciones en Arduino según convenga para que el robot ejecute las acciones demandadas.

6. RESULTADOS Y DISCUSIÓN.

6.1. Videos de demostración del funcionamiento.

Enlace a los videos en Google Drive:

https://drive.google.com/drive/folders/12wB_s3CoTHbYP0oDT6Me6KeXcVdq7wTe?usp=sharing

6.2. Discusión

A la vista de todas las pruebas anteriores se pueden sacar las siguientes conclusiones referentes a las fuentes de error de este brazo robótico:

- **Huelgos en los engranajes de los servomotores:** esta es una fuente de error que va a afectar en todos los casos a la precisión del robot. A parte, depende de la dirección de movimiento así que no se puede corregir en lazo abierto.

- **Deformaciones de los eslabones**: debido a su construcción en plástico rígido las deformaciones de los eslabones afectan en gran medida a la precisión, no afectando a la repetibilidad. En robots industriales se requieren materiales más rígidos para la construcción de un robot tales como metal, el problema surge en que se requerirían motores con más fuerza y más potentes para mover el peso extra.

- **Error en la librería de Arduino:** existe un error en la librería de Arduino que provoca que al dar los ángulos a cada servo no alcancen este valor si no que se queden en un ángulo anterior (mayor o menor en cada caso). Para corregir este error era necesario corregir las librerías, pero no merecía la pena porque el error que aportan otras partes era mayor que este error.

- La construcción en plástico de los **elementos que unen el engranaje del servomotor a su eslabón correspondiente** no es una decisión acertada (Figura 54 y Figura 53). Estos elementos soportan grandes esfuerzos cuando el servomotor ejecuta un movimiento y los dientes internos se acaban desgastando rápidamente. Esto provoca huelgos y mayores errores de precisión.

Figura 54 Conjunto eslabón base con servomotor.

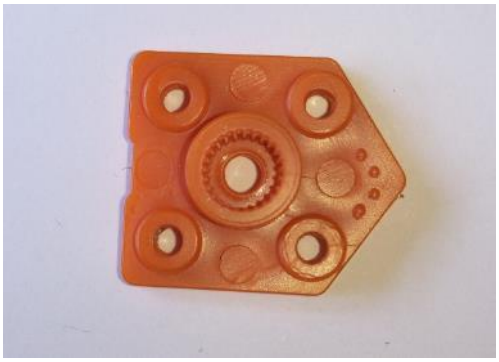


Figura 53 Detalle de la pieza de unión.

El robot también cuenta con unos puntos positivos que logran que sea bastante funcional. Estos puntos positivos son:

- **El uso de servomotores** de gran par consigue un dispositivo de gran sencillez (sin engranajes) pero que es capaz de hacer movimientos rápidos y de mantener su posición estable.
- El **diseño de la pinza** con cuadrilátero articulado y engranajes de la pinza es muy adecuado ya que logra bastante robustez y fuerza en el agarre.
- El uso de Arduino ha permitido implementar un **control mediante bluetooth** desde smartphone, con una interfaz muy sencilla y bastantes opciones disponibles para interactuar con el robot.

Características del brazo desde el punto de vista robótico:

- Este robot consigue una gran repetibilidad. Logra ser constante en el error cometido, lo cual se aprovecha en el algoritmo de corrección del error.
- La precisión es pobre, pero se ha podido corregir por software.
- Alta velocidad de movimiento. Aunque es muy sensible a movimientos rápido ya que aparecen vibraciones que pueden dañar a los motores o a la estructura.
- Gran capacidad de mantener posiciones estables, incluso en posiciones extremas. Sin embargo, la carga de trabajo útil se reduce cuando se estira el brazo. Esto limita a 150g el peso que puede levantar el robot.
- La accesibilidad está reducida debido a la gran longitud de la pinza. Esto se ha conseguido solventar mediante programación, pero perdiendo la capacidad de seleccionar la elevación con que acceder al punto.

6.3. Cumplimiento de objetivos.

El robot es capaz de alcanzar objetos en el plano con cierta precisión y transportar estos a distintas posiciones. Se han implementado algoritmos de mejora de la precisión, sin embargo, dado que solo hay posibilidad de actuar en lazo abierto, es imposible lograr una alta precisión.

Por el contrario, el robot sigue siendo impreciso y aparecen errores cuando se realizan ciclos de movimientos en bucle. La solución posible a este problema es la integración de un control en lazo cerrado realimentando la posición mediante el uso de sistemas de posicionamiento como Kinect.

7. CONCLUSIONES.

Se ha desarrollado la programación de Arduino destinada al control del brazo robótico Tinkerkit Braccio de Arduino. Concretamente se buscaba que el robot alcance objetos con la pinza con precisión.

Tras el análisis de la cinemática del robot se ha hecho un programa que posicione el robot mediante coordenadas cartesianas, añadiendo un cálculo de la elevación posible para alcanzar dicho punto.

Debido a las no linealidades tales como huelgos y deformaciones aparecían imprecisiones en la posición alcanzada. Se ha desarrollado un algoritmo de mejora de la precisión cuyo resultado es bueno, aunque no logra paliar del todo los problemas de posición.

La creación de la aplicación en Android mediante el sistema MIT App Inventor es sencilla y permite crear aplicaciones funcionales. La aplicación creada presenta un interfaz sencillo, aunque funcional con variedad de funciones. La creación del programa en Arduino y de la App ha tenido que hacerse de forma simultánea ya que se requiere una comunicación entre los dos dispositivos.

En conclusión, se ha logrado dotar de funcionalidad al brazo robótico partiendo del control angular de los servomotores, llegando a poder controlarlo mediante bluetooth con un smartphone con funciones tales como posicionamiento en coordenadas cartesianas, movimiento automático, movimiento de la pinza y variación de su velocidad.

8. LINEAS FUTURAS.

- Realizar un programa que permita visualizar la posición del robot en tiempo real, representando un dibujo con los movimientos de los eslabones.
- Control realimentado mediante control de posición. Por ejemplo, Kinect.
- Utilización de cámaras para detección de objetos y obtención de su posición para agarrarlos sin intervención del usuario.
- Diseño de una maqueta donde se utilice el robot junto a otros sensores (detectores de objetos, finales de carrera, etc.) para simular visualmente una cadena de producción.

9. BIBLIOGRAFÍA.

Ollero, A. (2001). *Robótica: manipuladores y robots móviles.* Universidad de Sevilla.

Wiki de Robótica (2016. *Clasificación de los robots*. Recuperado de: http://wiki.robotica.webs.upv.es/wiki-de-robotica/introduccion/clasificacion-de-robots/

Bluetooth, capa física y acceso al medio. Recuperado de: https://www.gta.ufrj.br/grad/09_1/versao-final/bluetooth/Page323.htm

Prometec. *El módulo bluetooth hc-05.* Recuperado de: https://www.prometec.net/bt-hc05/

HC Serial Bluetooth. *Products User Instructional Manual.* Recuperado de: https://cdn.makezine.com/uploads/2014/03/hc_hc-05-user-instructions-bluetooth.pdf

HC-05 - Bluetooth Module. Recuperado de: https://components101.com/wireless/hc-05-bluetooth-module

Bluetooth en Arduino. Recuperado de: https://aprendiendoarduino.wordpress.com/tag/hc-05/

Configuración del módulo bluetooth HC-05 por comandos AT. Recuperado de: http://digirobs.blogspot.com.es/2015/10/configuracion-del-modulo-bluetooth-hc.html

Arduino (2018). *Arduino.* Recuperado de: https://www.arduino.cc/

Guía de Iniciación a App Inventor (2015). Recuperado de: http://codeweek.eu/resources/spain/guia-iniciacion-app-inventor.pdf

GitHub (2018). *Braccio Library for Arduino.* Recuperado de: https://github.com/arduino-org/arduino-library-braccio

GitHub (2018). *Arduino libraries , Servo.h.* Recuperado de: https://github.com/arduino-libraries/Servo/blob/master/src/Servo.h

Brainy-bits (2015). *How to connect and use Analog Joystick with Arduino.* Recuperado de: https://www.brainy-bits.com/arduino-joystick-tutorial/

Luis Llamas (2018). *Controlar un Servo con Arduino.* Recuperado de: https://www.luisllamas.es/controlar-un-servo-con-arduino/