

TRABAJO FINAL DE GRADO

Grado en Ingeniería Electrónica Industrial y Automática

**CONTROL DE UN BRAZO ARTICULADO MEDIANTE
DISPOSITIVO MÓVIL**



Memoria i Anexos

Autores: Marc Ruiz Vinyeta
Ricard Llauger Roca

Director: Sebastián Tornil Sin

Convocatoria: Mayo 2017



Resum

Aquest projecte de final de grau consisteix en el disseny i construcció d'un braç robot articulat controlat mitjançant una aplicació mòbil.

A través d'aquesta aplicació es podrà tenir el control del braç i permetrà ensenyar uns punts determinats a l'espai tridimensional als quals posteriorment el seu extrem hi podrà accedir realitzant un determinat recorregut.

El robot, es diferencia de la resta de productes del mercat per tenir una finalitat educativa i un pressupost reduït. A més a més d'haver estat dissenyat amb l'objectiu d'utilitzar una tecnologia actual i optimitzar al màxim el seu pes.

Per a realitzar aquest projecte, s'ha fet un estudi de l'estat del art per tal de poder obtenir el coneixement de l'estat actual en el mercat d'aquest tipus de productes i així poder establir unes especificacions a complir.

Per últim, s'ha estudiat els entorns que permetran el disseny, el control i la realització d'una aplicació mòbil per aconseguir l'objectiu d'aquest treball.

Resumen

Este proyecto de final de grado consiste en el diseño y construcción de un brazo robot articulado controlado mediante una aplicación móvil.

A través de esta aplicación se podrá tener el control del brazo y permitirá enseñar unos puntos determinados en el espacio tridimensional a los que posteriormente su extremo podrá acceder realizando un determinado recorrido.

El robot, se diferencia del resto de productos del mercado por tener una finalidad educativa y un presupuesto reducido. A parte de haber estado diseñado con el objetivo de utilizar una tecnología actual y optimizar al máximo su peso.

Para realizar este proyecto, se ha hecho un estudio del estado del arte con la finalidad de poder obtener el conocimiento del estado actual en el mercado de este tipo de productos y así poder establecer unas especificaciones a cumplir.

Por último, se ha realizado un estudio de los entornos que permiten el diseño, el control y la realización de una aplicación móvil para conseguir el objetivo de este trabajo.

Abstract

This Project consists of designing and manufacturing a robotic arm controlled by a mobile application.

Through this app it will be possible to take control of the arm and it will allow teaching different points in the 3D space which could be accessible by the extreme making a certain route.

The robot differs from the other products in the market due to its educational purpose with a reduced budget. In addition, it has been designed with the objective of using the latest technology and optimising its weight.

To carry out this project, a state of the art study has been done to obtain the knowledge of the current state of the market for these kind of products and in this way set the specifications that the arm has to satisfy.

To conclude, a study has been made about the different software that can give the opportunity to achieve the proposal of this project.

Agradecimientos

Los autores de este proyecto quieren agradecer en primer lugar el soporte recibido por parte del tutor, Sebastián Tornil, por guiar, asesorar, y por su profesionalidad en el mundo de la docencia. Sin su ayuda, este proyecto de final de grado se hubiera hecho mucho más complicado.

En segundo lugar agradecer a sus familias el soporte recibido en los momentos más difíciles y por estar a su lado durante los años de sus estudios. Sin ellos, nada de esto hubiera sido posible.

En tercer lugar, agradecer a todas aquellas personas que han dado soporte a este proyecto de manera indirecta. Amigos, compañeros de trabajo y toda la gente que ha abierto la puerta de sus conocimientos y los ha compartido con los autores.

Por último, agradecer la facilidad de trabajo entre los autores que aparte de reforzar una gran amistad que se inició al principio de sus estudios universitarios ha favorecido a la creación de un bonito equipo de trabajo con grandes aspiraciones.

Índice

Resum	1
Resumen.....	2
Abstract	3
Agradecimientos.....	4
1. Introducción	19
1.1. Origen del trabajo.....	19
1.2. Motivación	19
1.3. Requerimientos previos.....	19
2. Objetivos	22
2.1. Objetivos del trabajo	22
2.2. Alcance del trabajo	23
3. Antecedentes	25
3.1. Configuración mecánica del producto	25
3.2. Programación de robots	27
3.2.1. Programación por guiado.....	27
3.2.2. Programación textual.....	27
3.3. Presencia en el mercado	28
4. Especificaciones	33
4.1. Características genéricas	33
4.2. Características técnicas.....	34
5. Diseño mecánico	37
5.1. Configuración global.....	37

5.1.1.	Principio de diseño	37
5.1.2.	La transmisión.....	38
5.1.3.	Elementos mecánicos.....	39
5.2.	Diseño de las articulaciones.....	42
5.2.1.	Pinza.....	43
5.2.2.	Muñeca	44
5.2.3.	Codo.....	47
5.2.4.	Hombro.....	49
5.2.5.	Cintura	50
5.2.6.	Barras estructurales.....	52
5.3.	Evaluación del conjunto.....	54
5.3.1.	Cálculos estáticos sobre articulaciones.....	54
5.3.2.	Análisis estático	55
5.3.3.	Análisis dinámico	62
5.3.4.	Dimensionado de los pares mínimos de los motores o transmisiones.	69
6.	Cinemática del brazo articulado	71
6.1.	Cálculo del modelo cinemático directo	72
6.2.	Cálculo del modelo cinemático inverso.....	74
7.	Diseño electrónico.....	78
7.1.	Selección de componentes.....	78
7.1.1.	Placas de control.....	78
7.1.2.	Servomotores	82
7.1.3.	Bluetooth	87

7.1.4.	Android	90
7.2.	Componentes electrónicos del brazo articulado	91
7.3.	Esquema de conexión.....	92
8.	Protocolo de comunicaciones.....	96
9.	Programación del controlador del brazo robot.....	101
9.1.	Diagrama de flujo del programa.....	101
9.2.	Programación con Arduino.....	104
9.3.	Función cálculo movimiento	108
9.4.	Desarrollo del código	108
10.	Programación y diseño de la aplicación para dispositivo móvil.....	110
10.1.	Android Studio	110
10.1.1.	Estructura del proyecto	110
10.1.2.	Interfaz de usuario	111
10.1.3.	Lenguaje de programación.....	113
10.2.	Diseño de la aplicación	113
10.2.1.	Menú principal	114
10.2.2.	Vinculación robot	115
10.2.3.	TEACH	116
10.2.4.	EXECUTOR.....	119
10.3.	Simulaciones	122
10.3.1.	Genymotion.....	122
10.3.2.	Smartphone	124
10.4.	Programa Android Studio	130

11.	Construcción del prototipo	132
11.1.	Elementos del brazo articulado	132
11.1.1.	Base.....	132
11.1.2.	Eslabones	134
11.2.	Brazo articulado	138
12.	Análisis del impacto ambiental	141
13.	Planificación temporal	143
14.	Estudio económico.....	145
14.1.	Costes de desarrollo y prototipado	145
14.2.	Coste de venta al público.....	147
15.	Conclusiones	152
16.	Bibliografía	155
	Anexo A: Planos del prototipo.....	160
	PLANO 1. BRAZO UNIÓN HOMBRO-CODO	167
	PLANO 2. BRAZO UNIÓN CODO-MUÑECA	169
	PLANO 3. BASE BRAZO ROBÓTICO.....	171
	PLANO 4. EJES ARTICULACIONES	173
	PLANO 5. EJES ESTRUCTURALES Y DE POLEAS TENSORAS	175
	PLANO 6. POLEA DE 24 DIENTES.....	177
	PLANO 7. POLEA DE 36 DIENTES.....	179
	PLANO 8. POLEA DE 20 DIENTES.....	181
	Anexo B: Código Arduino.....	184
	Anexo C: Esquema electrónico.....	221

Anexo D: Código Android Studio.....222

Índice de ilustraciones

Ilustración 1. Esquema del objetivo del proyecto	22
Ilustración 2. Tinkerkit Braccio Arduino Robotic Arm	29
Ilustración 3. Ejemplos de aplicaciones Braccio	30
Ilustración 4. Kit de adquisición del Braccio	31
Ilustración 5. Esquema del diseño del brazo articulado.....	33
Ilustración 6. Rodamientos de agujas KH08-10.....	39
Ilustración 7. Cojinetes F8-16M	40
Ilustración 8. Anillos de fijación	41
Ilustración 9. Anillo de fijación KFL08	41
Ilustración 10. Articulación de la base del prototipo.....	42
Ilustración 11. Pinza del prototipo.....	43
Ilustración 12. Configuración de la muñeca del prototipo.....	44
Ilustración 13. Diseño pinza prototipo	45
Ilustración 14. Fijación de la pinza al eje(I)	46
Ilustración 15. Fijación de la pinza al eje (II)	46
Ilustración 16. Fijación de la pinza al eje (III)	47
Ilustración 17. Diseño final muñeca.....	47
Ilustración 18. Fijación de la articulación del codo al eje(I).....	48
Ilustración 19. Fijación de la articulación del codo al eje(II).....	48
Ilustración 20. Fijación de la articulación del hombro al eje(I).....	50
Ilustración 21. Configuración de la base (I)	51
Ilustración 22. Configuración de la base (II).....	51

Ilustración 23. Configuración de la base (III)	52
Ilustración 24. Barras estructurales.....	53
Ilustración 25. Configuración crítica	54
Ilustración 26. Análisis estático pinza (I).....	56
Ilustración 27. Análisis estático pinza (II).....	57
Ilustración 28. Análisis estático pinza (III).....	57
Ilustración 29. Análisis estático pinza (IV)	58
Ilustración 30. Análisis estático pinza (V)	58
Ilustración 31. Análisis estático muñeca	59
Ilustración 32. Análisis estático codo.....	60
Ilustración 33. Análisis estático hombro	61
Ilustración 34. Análisis dinámico pinza.....	63
Ilustración 35. Análisis dinámico muñeca	64
Ilustración 36. Análisis dinámico codo	65
Ilustración 37. Análisis dinámico hombro	66
Ilustración 38. Análisis dinámico base.....	68
Ilustración 39. Geometría del brazo robot para cálculos de cinemática	71
Ilustración 40. Relaciones geométricas- cinemática directa.....	74
Ilustración 41. Modelo cinemático inverso.....	75
Ilustración 42. Relaciones geométricas-cinemática inversa	75
Ilustración 43. IDE Arduino	81
Ilustración 44. Arduino Mega 2560	82
Ilustración 45. Ejemplo servomotor	83

Ilustración 46. Elementos principales de un servomotor.....	83
Ilustración 47. Señal PWM para el control del servomotor	84
Ilustración 48. Sistema de control interno del servomotor	85
Ilustración 49. Selección servomotores.....	86
Ilustración 50 y 51. TowerPro MG995 y MG958	86
Ilustración 52. Tower Pro MG959.....	87
Ilustración 53. Logotipo Bluetooth	88
Ilustración 54. Piconet.....	88
Ilustración 55. Módulo Bluetooth HC-06.....	89
Ilustración 56. Patillaje conexión módulo HC-06.....	90
Ilustración 57. Logotipo Android	90
Ilustración 58. Esquema de conexión electrónico.....	93
Ilustración 59. Esquema movimiento JOINT.....	96
Ilustración 60. Esquema movimiento X, Y, Z	97
Ilustración 61. Diagrama de funcionamiento del programa de Arduino	102
Ilustración 62. Generación de señal PWM en las salidas digitales de Arduino.....	105
Ilustración 63. Estructura proyecto Android Studio	111
Ilustración 64. Interfaz de usuario Android Studio.....	112
Ilustración 65. Pantalla menú aplicación	114
Ilustración 66. Pantalla vinculación aplicación	115
Ilustración 67. Pantalla JOINT aplicación	117
Ilustración 68. Pantalla X,Y,Z aplicación.....	119
Ilustración 69. Pantalla executor aplicación	120

Ilustración 70. Simulador Genymotion (I)	122
Ilustración 71. Simulador Genymotion (II)	123
Ilustración 72. Simulador Genymotion (III)	123
Ilustración 73. Simulador Genymotion (IV)	124
Ilustración 74 y 75. Ejecución real aplicación 1 y 2	125
Ilustración 76 y 77. Ejecución real aplicación 3 y 4	126
Ilustración 78 y 79. Ejecución real aplicación 5 y 6	127
Ilustración 80 y 81. Ejecución real aplicación 7 y 8	128
Ilustración 82 y 83. Ejecución real aplicación 9 y 10	129
Ilustración 84. Ejecución real aplicación 11.....	130
Ilustración 85. Construcción de la base(I)	132
Ilustración 86. Construcción de la base (II).....	132
Ilustración 87. Preparación servomotores	133
Ilustración 88. Construcción de la estructura de la base	133
Ilustración 89. Base del prototipo	134
Ilustración 90. Construcción eslabones del prototipo	134
Ilustración 91. Articulación del hombro del prototipo.....	135
Ilustración 92. Articulación del codo del prototipo.....	135
Ilustración 93. Articulación muñeca del prorotipo.....	136
Ilustración 94. Aspecto del brazo articulado	136
Ilustración 95. Montaje pinza	137
Ilustración 96. Montaje estructura del brazo.....	137
Ilustración 97. Aspecto final del brazo articulado (I).....	138

Ilustración 98. Aspecto final del brazo articulado (II).....	139
Ilustración 99 y 100: Aspecto final del brazo articulado(I) y (II).....	153
Ilustración 101. Geometría de las poleas dentadas de paso T5.....	161
Ilustración 102: Geometría de los rodamientos de agujas HK y BK.....	162
Ilustración 103. Soporte KFL 08 acotado.....	164
Ilustración 104. Geometría rodamientos de empuje serie Fx.....	164
Ilustración 105. Esquema de conexión electrónico.....	221

Índice de tablas

Tabla 1. Configuraciones de los brazos robóticos	26
Tabla 2. Ejemplos de brazos robóticos en el mercado (I).....	28
Tabla 3. Ejemplos de brazos robóticos en el mercado (II).....	29
Tabla 4. Tabla de características del Braccio	30
Tabla 5. Modos de control del brazo articulado.....	34
Tabla 6. Especificaciones técnicas del prototipo (I).....	34
Tabla 7. Especificaciones técnicas del prototipo (II).....	34
Tabla 8. Especificaciones técnicas del prototipo (III).....	35
Tabla 9. Par de sollicitación estática	61
Tabla 10. Par de sollicitación dinámica.....	68
Tabla 11. Par de diseño	69
Tabla 12. Par de diseño con factor de seguridad	69
Tabla 13. Valores de la representación de Denavit-Hartenberg.....	73
Tabla 14. Comparativa Arduino Uno VS Raspberry Pi (B)	79
Tabla 15 y 16. Tabla características TowerPro MG995 y MG958	86
Tabla 17. Tabla características TowerPro MG959.....	87
Tabla 18. Paquetes de datos pantalla teaching.....	99
Tabla 19. Paquetes de datos pantalla executor	99
Tabla 20. Paquetes de datos pinza	100
Tabla 21. Variables Arduino	105
Tabla 22. Funciones básicas Arduino	106
Tabla 23. Costes de ingeniería	145

Tabla 24. Costes de materiales	146
Tabla 25. Coste total del desarrollo y prototipado.....	147
Tabla 26. Costes de materiales	147
Tabla 27. Geometrías de las correas normalizadas de métrica T.....	160
Tabla 28. Valores geométricos según número de dientes en poleas con correas de 10 mm	161
Tabla 29. Valores geométricos según número de dientes en poleas con correas de 16 mm	162
Tabla 30. Valores geométricos de los rodamientos de agujas HK y BK.....	163
Tabla 31. Valores geométricos de los rodamientos de empuje serie Fx.....	165
Tabla 32. Valores de los tornillos según su métrica.....	165
Tabla 33. Valores de las tuercas según su métrica	165
Tabla 34. Valores para los ejes estructurales del plano Num. 5. (SOPORTES ESTRUCTURA Y EJES DE POLEAS TENSORAS)	166
Tabla 35. Valores para los ejes de las articulaciones, acompaña al plano Núm. 4	166



1. Introducción

El presente Trabajo de Final de Grado (TFG), tiene como objetivo el diseño mecánico y electrónico de un brazo robot articulado programable controlado mediante la placa de programación Arduino y una aplicación de dispositivo móvil. Se trata de tener el control del brazo articulado mediante una aplicación que nos permitirá definir por un lado el movimiento de cada articulación y por otro la posición de su pinza, para posteriormente programar un recorrido concreto.

1.1. Origen del trabajo

Ambos autores del documento han compartido desde el inicio de su vida universitaria vivencias y amistad que han causado que en el fin de su etapa como estudiantes de grado hayan decidido poner en común los conocimientos obtenidos.

1.2. Motivación

Las ganas de “crear” y aprender a diseñar aplicaciones para dispositivos móviles en un momento en que todo se rige por esta vía de intercambio de información, ha hecho que este proyecto se haga realidad.

Adicionalmente, la afición a la programación de microcontroladores y la admiración hacia el diseño de automatismos y a la robótica que se ha visto incrementado a medida que han ido avanzando los cursos y se han ido impartiendo las asignaturas durante el grado, han supuesto elementos clave en la elección de la temática del proyecto.

Cabe destacar que el contacto con el brazo robot del laboratorio de robótica de la Universidad también ha sido otro de los ingredientes esenciales que ha hecho decantar a los autores por esta rama.

1.3. Requerimientos previos

Los requerimientos necesarios para la realización del proyecto son los obtenidos durante la realización de las asignaturas del grado que están más encarradas al objetivo.

Por la parte del diseño mecánico, las asignaturas que tienen más peso en cuanto a contenido y conocimientos utilizados en el proyecto son: Diseño de máquinas y mecanismos, Resistencia y Elasticidad de los materiales, Estructuras, Procesos de fabricación y Expresión Gráfica.

Por otro lado, en cuanto a la parte electrónica, las asignaturas con más influencia son: Informática, Control Industrial y Automatización, Robótica Industrial e Informática Industrial.

En este caso, no se tenía ningún conocimiento previo en referencia al diseño de aplicaciones y por tanto, más que en otros aspectos del trabajo, en este ámbito el aprendizaje autónomo jugará un papel mucho más importante.

2. Objetivos

El objetivo principal de este proyecto es el diseño y la construcción de un brazo robot manipulador programable mediante un dispositivo móvil (ver ilustración 1).



Ilustración 1. Esquema del objetivo del proyecto

2.1. Objetivos del trabajo

Con el propósito de conseguir el objetivo principal de dicho proyecto, los autores deberán, a la vez, ir completando los siguientes objetivos que les permitirá alcanzar su finalidad:

- Diseño mecánico.
- Modelado cinemático.
- Selección de componentes.
- Programación de la placa de control.
- Programación de la aplicación para dispositivo móvil.
- Construcción del prototipo.
- Realización de pruebas de funcionamiento.
- Elaboración de la documentación.

El brazo robot estará diseñado previamente mediante el entorno *SolidWorks*. Cabe destacar que el diseño se basará a partir de dos limitaciones: el peso que debe ser capaz de sostener en la pinza y los elementos mecánicos y electrónicos que se pueden encontrar normalizados en el mercado y a los que se puede tener acceso. En cuanto el material del que estará hecha la estructura del brazo será el aluminio.

Deberán realizarse todos los cálculos necesarios con la finalidad de seleccionar los elementos de transmisión mecánicos (poleas y correas) y los electrónicos (servomotores) los cuales serán controlados mediante la placa Arduino Mega 2560.

El control del brazo se realizará mediante una aplicación móvil que a través de un módulo de comunicación *Bluetooth* se comunicará con Arduino y transmitirá la información necesaria de manera inalámbrica.

El brazo robot diseñado tiene una finalidad docente para que en un futuro no muy lejano pueda ayudar a alumnos a aprender a establecer comunicaciones inalámbricas y al diseño de aplicaciones móvil.

2.2. Alcance del trabajo

El proyecto constará de una previa planificación temporal con fechas límite por parte de los dos integrantes para el cumplimiento de las tareas asignadas y así de este modo se creará la posibilidad de poder trabajar en paralelo en la realización de determinados ejercicios.

Inicialmente se realizará el diseño y cálculo de componentes con el fin de cumplir los objetivos anteriormente expuestos teniendo en cuenta la oferta de mercado. Una vez concluida esta tarea se procederá a la adquisición de todos los elementos tanto mecánicos como electrónicos necesarios para realizar el montaje del prototipo.

Una vez reunidos todos los componentes se procederá al ensamblaje.

Paralelamente se realizará un estudio teórico del estado del arte de los brazos robóticos, de su principio de funcionamiento y los componentes que los forman. También se realizará un estudio teórico de la placa Arduino y su entorno.

A partir de aquí se iniciará la elaboración del programa de control del microcontrolador, el diseño de la aplicación y la comunicación entre ambos.

Por último se realizará un análisis del impacto ambiental del proyecto y un estudio económico.



3. Antecedentes

Antes de empezar a abordar el desarrollo del brazo robótico, hay que realizar cierta formación en cuanto al concepto y el principio de funcionamiento de este tipo de aparatos. Una vez alcanzada una comprensión general del producto, se procurará centrar los esfuerzos en analizar globalmente el estado del arte en que se encuentran los brazos robóticos de juguete y los componentes que los integran.

3.1. Configuración mecánica del producto

En la actualidad, un brazo robótico se define como un tipo de brazo mecánico, normalmente programable, con funciones similares a las de un brazo humano.

Los brazos robóticos se componen de actuadores que pueden permitir o bien un movimiento rotacional o bien uno lineal; todos ellos unidos forman una cadena cinemática. En términos populares, cuando se hace referencia a los aparatos que conforman el mecanismo, se suele hablar o de articulaciones (siguiendo la analogía con el brazo humano) o bien de grados de libertad.



Por lo que respecta a la articulación más alejada de la base del brazo, ésta suele recibir el nombre de *end effector*. Posiblemente es la parte más interesante y donde se plasma la razón de ser del diseño robótico. El extremo puede adquirir formas complejas tales como la reproducción de una mano humana o bien adquirir formas más sencillas pero no menos funcionales tales como una pinza o un imán.

Según cómo se distribuya la cadena cinemática compuesta por las anteriores articulaciones, el principio mecánico del brazo robótico y por tanto su espacio accesible, puede cambiar drásticamente.

En la tabla 1 se procura ilustrar las configuraciones de brazos robóticos más integradas y con más apoyo técnico. Las imágenes van acompañadas de diagramas y la envoltura de posiciones accesibles que resulta.

Así mismo, también se aporta información adicional en forma de comentarios citando sus características más potentes y algunas aplicaciones industriales para terminar de comprender la integración de su principio de funcionamiento a la realidad.

Tabla 1. Configuraciones de los brazos robóticos

Geometría	Diagrama del solido libre	Espacio accesible	Ejemplo	Comentarios
Cartesiana 				<p>Simplicidad, alta precisión y rigidez.</p> <p>Aplicaciones: -Maquinas CNC -Impresión 3D</p>
Cilíndrica 				<p>Simplicidad, alta precisión y rigidez.</p> <p>Aplicaciones: -Traslación horizontal de piezas (Pick& Place)</p>
Esférica 				<p>Primer brazo con capacidad para trabajar en planos inclinados.</p> <p>Aplicaciones: Soldaduras por puntos. Manipulación de partes y fundiciones en cadenas de montaje.</p>
Angular o antropomórfica 				<p>Versión más completa de brazo robótico.</p> <p>Aplicaciones: Paletizado. Soldadura por arco y puntos, pintura etc.</p>
Tipo SCARA 				<p>Versión mejorada del diseño cilíndrico, mayor espacio interior.</p> <p>Aplicaciones: -Traslación horizontal de piezas (Pick& Place)</p>
Paralela 				<p>El end effector es la misma base a la cual se pueden acomodar los elementos.</p> <p>Aplicaciones: Simulaciones de vuelo. Micro manipuladores.</p>

Gracias al anterior sumario de configuraciones de brazos robóticos, se ha conseguido aportar una perspectiva global de cómo ha evolucionado el diseño de estos aparatos y poner un poco de luz sobre cómo enfocar el desarrollo del producto a diseñar en este proyecto.

Cabe decir, que es importante no perder la esencia de lo que se aborda en este proyecto. El brazo a desarrollar será un **prototipo de brazo industrial**, y en este apartado se han plasmado robots destinados a la industria, hay que valorar ante todo la capacidad de juego y su versatilidad

3.2. Programación de robots

La programación de un robot consiste en la introducción de las órdenes necesarias para que éste realice el conjunto de tareas para las que ha sido diseñado. Con esta finalidad existen varios procedimientos de programación: la programación guiada y la programación textual aunque casi siempre se usa su combinación.

3.2.1. Programación por guiado

En este tipo de programación el operario o usuario que maneja el robot, lo guía de manera manual y prepara al brazo articulado para que sea capaz de realizar de manera independiente una trayectoria.

Cada uno de los movimientos que la persona le indica al robot se van almacenando con la finalidad de poder ser repetidos posteriormente. De este modo, se hablará de una parte de “*TEACH*” dónde el usuario realiza de manera manual los movimientos y configura el brazo para que guarde posiciones determinadas. Y por otro lado, se hablará del “*EXECUTOR*” dónde el usuario se limitará a pedir al robot que se desplace siguiendo una trayectoria determinada a través de los puntos que han sido almacenados anteriormente.

En la programación guiada, es necesario tener el prototipo del robot para la elaboración del programa. En la mayoría de ocasiones se utiliza un sistema de guiado en forma de *joystick* para mover las articulaciones del robot.

3.2.2. Programación textual

Este tipo de programación se lleva a cabo a través de un lenguaje informático. El programa consiste en un texto formado por un conjunto de instrucciones. Cuando el programa es grabado en la memoria del robot, éste realiza las instrucciones.

La programación textual permite realizar operaciones más complejas y con mayor grado de precisión. Además, a diferencia de la programación guiada, en este caso no es necesaria la presencia del robot para realizar el programa.

3.3. Presencia en el mercado

En el siglo XXI, la era digital, resulta cada vez más difícil encontrar herramientas u objetos que no incorporen algún sistema electrónico, por muy simple que sea. El sector del ocio es un ejemplo claro de esta tendencia; cada vez se popularizan más los juguetes por control remoto como los vehículos, los robots o los drones.

En cuanto la materia de interés del proyecto, el brazo robótico tiene una presencia muy potente en el sector industrial. Se presenta como un elemento rápido, preciso y versátil capaz de incorporarse en las cadenas de producción de manera excelente.

Sin embargo, el brazo robot que se pretende diseñar en este trabajo tendrá una finalidad docente debido a que es la opción más económica en frente a la adquisición de un robot industrial.

Tabla 2. Ejemplos de brazos robóticos en el mercado (I)





OWI-535 Robotic Arm Edge		Lynxmotion AL5A	
			
Precio	46.69€	Precio	292€
Peso	0767 kg	Peso	0.652 kg
Alcance vertical	0.381 m	Alcance vertical	0.4 m
Alcance horizontal	0.32 m	Alcance horizontal	0.2 m
Articulaciones	5	Articulaciones	5
Máxima carga	0.1 kg	Máxima carga	0.126 kg
Programable	No	Programable	Si
Modo de control	Control Remoto	Modo de control	Ordenador

Tabla 3. Ejemplos de brazos robóticos en el mercado (II)

Satelite S300100		RobotGeek Snapper	
			
Precio	35.95€	Precio	184.06€
Peso	0.603 kg	Peso	0.750 kg
Alcance vertical	0.368 m	Alcance vertical	0.387 m
Alcance horizontal	0.279 m	Alcance	0.29 m
Articulaciones	5	Articulaciones	5
Máxima carga	0.085 kg	Máxima carga	0.050 kg
Programable	Si	Programable	Si
Modo de control	Ordenador	Modo de	Control remoto

Como se puede observar en las tablas 2 y 3, en el momento en el que se intenta añadir al dispositivo cierta capacidad de programación o componentes más robustos, los precios se disparan.

No obstante, es interesante recopilar las características técnicas que configuran los productos más populares ya que servirán como base de referencia a la hora de designar las solicitudes del brazo robótico a diseñar durante el proyecto.

Debido a que la placa de control que se ha decidido utilizar en el trabajo es el Arduino Mega 2560 se cree necesario hacer referencia al brazo robótico de la misma marca: *Tinkerkit Braccio Arduino Robotic Arm* (ilustración 2).



Ilustración 2. Tinkerkit Braccio Arduino Robotic Arm

Tal y como el prototipo que se creará en este proyecto, también es un brazo robótico controlado por una placa Arduino. Está diseñado para uso de “escritorio” y viene con un Kit que ofrece versatilidad a la hora de su montaje ya que se puede ensamblar según requerimientos y funciones.

A continuación se adjunta su tabla de características:

Tabla 4. Tabla de características del Braccio

Alcance horizontal:	80 cm
Alcance vertical:	52 cm
Anchura de la base:	14 cm
Anchura de la pinza:	9 cm
Longitud del cable:	40 cm
Máxima capacidad de carga:	150 g
Peso máximo de la base:	400 g
Servomotores:	2 x SR 311, 4 x SR431
Peso total:	0.792 kg
Voltaje de funcionamiento:	5 V
Consumo:	20 mW
Máxima corriente:	1.1 A

Su precio está alrededor de los 250 € y gracias a su característica principal, la flexibilidad y capacidad de adaptación a infinitos tipos de aplicaciones, se puede adaptar su extremo para poder soportar objetos tal y como por ejemplo, podrían ser cámara de fotos, Smartphones e incluso paneles solares.



Ilustración 3. Ejemplos de aplicaciones Braccio

El Kit que se adquiere en la compra de este producto ofrece una aspecto tal y como se muestra en la siguiente ilustración:

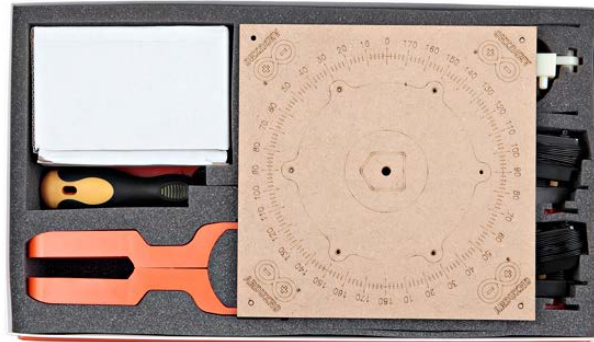


Ilustración 4. Kit de adquisición del Braccio

Un aspecto común de los brazos analizados es por un lado el uso de componentes electrónicos de modelismo como servomotores, potenciómetros y botones. Por otro lado, aquellos brazos que ofrecen capacidad de programación admiten la incorporación de placas de control.

Ya por último, comentar que la estructura articulada se presenta como la más atractiva de todas y también es la que cuenta con más antecedentes. Incluso, se han encontrado modelos antiguos ya fuera del mercado.

Configuraciones mencionadas en el apartado anterior como la SCARA y la paralela merecen también una mención, ya que se han detectado modelos capaces de realizar dibujos sobre papel y con cierta capacidad de juego.



4. Especificaciones

Debido a la complejidad del producto que se procede a desarrollar, es necesario empezar a fijar parámetros que el brazo deberá alcanzar para hacer posible su diseño. El número de variables a definir es muy elevado y casi imposible de abordar sino se imponen objetivos. Con la inspiración de los brazos robóticos analizados, se ha procurado proyectar un brazo robótico con alcances de juego diferentes a los demás pero a la vez sin alejarse demasiado de las propiedades mecánicas popularizadas.

4.1. Características genéricas

El producto a desarrollar se trata de un brazo robótico de carácter docente.

Ante esta última consideración, hay que entender pues, que se tratará de un brazo con una potencia y dimensiones reducidas al estar obligado más que nunca a no perjudicar la integridad del usuario.

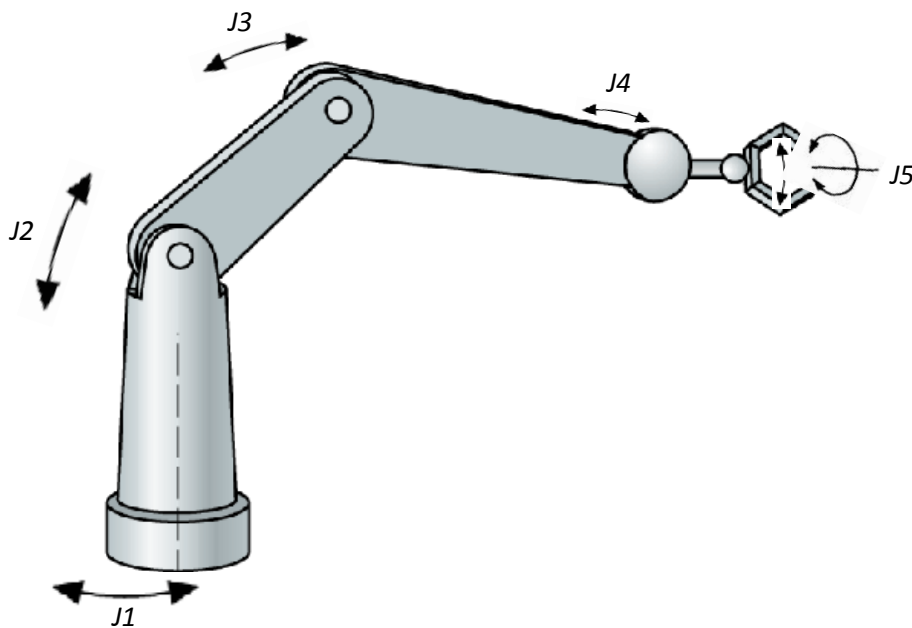


Ilustración 5. Esquema del diseño del brazo articulado

En cuanto al diseño, el brazo robótico articulado estará compuesto por un total de 5 articulaciones. La primera se comportará como base (J1) permitiendo la rotación horizontal de la estructura, mientras que la segunda (J2), la tercera (J3) y la cuarta (J4) conformarán el hombro, codo y muñeca del brazo siguiendo un símil con la extremidad humana. La última articulación será la que permitirá un movimiento rotacional de la pinza.

En cuanto al principio de funcionamiento, se diferenciarán dos modos de control presentados en la siguiente tabla:

Tabla 5. Modos de control del brazo articulado

Control Manual	El usuario será capaz de mover el brazo mediante una aplicación móvil con las interfaces básicas de un brazo robótico: jogging, que estará formado por dos tipos de movimientos (JOINT y X, Y, Z) y teaching. Además existirá la posibilidad de memorizar posiciones.
Control Automático	Si se dispone de posiciones memorizadas en el sistema durante el control manual, el control automatizado permite mover el brazo entre posiciones memorizadas de manera automática.

La comunicación entre el usuario y el brazo se podrá realizar mediante una aplicación móvil conectada por Bluetooth al microcontrolador Arduino, o bien directamente desde la consola que nos facilita la misma interfaz de Arduino.

4.2. Características técnicas

En cuanto a las especificaciones técnicas que se desean incorporar al diseño del brazo, en las tablas 6, 7 y 8 se resumen tanto aspectos básicos del diseño como valores cuantitativos que se desean alcanzar:

Tabla 6. Especificaciones técnicas del prototipo (I)

Alcance horizontal	0,45 m
Alcance vertical	0.60 m
Alcance angular de la base	180º
Alcance angular de las articulaciones	>90º
Carga máxima	0,1 kg

Tabla 7. Especificaciones técnicas del prototipo (II)

Velocidad angular base	J1	60º/s	1.047 rad/s
Velocidad angular hombro	J2	60º/s	1.047 rad/s
Velocidad angular codo	J3	90º/s	1.571 rad/s
Velocidad angular muñeca	J4	120º/s	2.094 rad/s
Velocidad angular giro muñeca	J5	360º/s	6.283 rad/s

Tabla 8. Especificaciones técnicas del prototipo (III)

Aceleración angular base	A1	600°/s	10.472 rad/s ²
Aceleración angular hombro	A2	600°/s	10.472 rad/s ²
Aceleración angular codo	A3	900°/s	15.708 rad/s ²
Aceleración angular muñeca	A4	1200°/s	20.944 rad/s ²
Aceleración angular giro muñeca	A5	1800°/s	64.832 rad/s ²



5. Diseño mecánico

5.1. Configuración global

El brazo robótico constará de una base la cual podrá ser considerada un empotramiento, pues será resistente y a la vez responsable de evitar el vuelco de la estructura. Ésta misma base tendrá la capacidad de girar entorno su eje vertical con la ayuda de un primer servomotor.

La base a la vez de ser la estructura también incorporará internamente el control del brazo por lo que ésta será hueca. Por otro lado al querer centrar las máximas masas posibles en la base, se situarán el máximo de actuadores en ella. La base sujetará mediante agujeros el eje de giro de la siguiente articulación, el hombro.

El material con el cual se realizará el brazo es una decisión muy importante, pues los pesos tanto de las barras como de todos los elementos serán decisivos para el diseño. Se debe considerar que la estructura debe ser de un material resistente pero a la vez ligero ya que deberá soportar los esfuerzos pero también los incrementará debido a su peso, es por ello que el material principal de la estructura será el aluminio, un material dúctil y resistente a la vez que ligero.

Este material con una densidad 3 veces más baja que la del acero es más que suficiente para el diseño. Es un material accesible y de fácil mecanización, por lo tanto una buena elección.

5.1.1. Principio de diseño

El principio mecánico a partir del cual se transmitirá el movimiento en las articulaciones es clave a la hora de diseñar la totalidad del brazo. Como se observará durante el desarrollo del proyecto, las dificultades técnicas son considerables y el diseño de todo el conjunto siempre se encuentra restringido a las decisiones que se tomen.

Por otro lado se debe tener en cuenta el punto de partida del diseño, pues sería un gran error empezar por la base. El diseño de un brazo robótico viene a ser un pequeño símil a un voladizo, no se puede diseñar un empotramiento sin saber qué esfuerzos debe resistir, por lo tanto se debe empezar por el punto más alejado de éste.

Un matiz muy importante el cual se debe remarcar antes de empezar a realizar el diseño, es que cómo todo diseño es un procedimiento iterativo. Esto significa que aunque se describirá de forma separada el diseño y los cálculos o análisis, estos están estrictamente relacionados, y como se ha dicho, al ser un procedimiento iterativo, se han realizado varios análisis completos anteriormente con los cuales se ha llegado al resultado mostrado en los apartados siguientes.

Esta mención hace especial referencia a que se colocarán ciertos elementos de transmisión los cuales han debido de ser calculados conjuntamente con el diseño a medida que se va realizando, para así asegurar su correcto funcionamiento.

5.1.2. La transmisión

En primer lugar se ha decidido aplicar el par únicamente a un lateral de cada articulación, ésta configuración se comentará más adelante por un posible inconveniente que puede provocar.

La transmisión de los pares aportados por los servomotores se realizará mediante un contacto directo entre los servomotores y las articulaciones donde se aplica el movimiento, en el caso del cierre de la pinza y del giro de la muñeca, mientras que en las demás articulaciones la transmisión se realizará mediante correas y poleas dentadas, en donde las poleas que transmiten el movimiento estarán directamente conectadas a la estructura de la articulación.

Los motivos por los cuales se ha decidido realizarlo de esta manera se exponen a continuación:

La configuración por contacto directo se situará en las partes del brazo robótico que son especialmente diseñadas para servomotores y que se encuentran en el mercado. Estas piezas son perfectas debido a la complejidad que tiene diseñar esta parte del brazo si se pretende realizar el giro de muñeca para poder coger cualquier objeto que este colocado en cualquier ángulo. Las configuraciones por poleas y correas se encuentran en las partes de diseño donde el eje motriz y el eje de salida son paralelos y que se encuentran a cierta distancia

El hecho de haber tomado la decisión de configurar la articulación del conjunto mediante contacto directo entre la polea y un lado del brazo, hace que el lado donde se realiza la interacción perciba mayores esfuerzos y se pueda desestabilizar. Se precisa pues, dotar el brazo de una distribución más simétrica de los esfuerzos de rotación. Debido a esta asimetría es necesario acomodar algún componente que asegure el paralelismo, estabilidad y buena transmisión del movimiento.

Para resolver este problema se han colocado varios elementos como son barras estabilizadoras entre ambos lados del brazo para corregir progresivamente la desviación que pueda suponer esta asimetría. Por otro lado se ha realizado la transmisión de manera que cada articulación vaya alternando el lado de aplicación de la fuerza, para intentar corregir una desviación con la siguiente transmisión.

5.1.3. Elementos mecánicos

Elementos de unión con movimientos libres

Se comprende con anterioridad que todos los elementos deben tener una posición en la que su funcionamiento sea el deseado, para cumplir este objetivo se deben tener en cuenta elementos tanto de fijación como de separación entre movimientos como el de evitar las interferencias entre los elementos rotacionales. Para solucionar este problema el diseño constará de varios tipos de rodamientos y fijaciones para conseguir este objetivo.

En concreto hay dos tipos de movimientos que se deben considerar para evitar interferencias y reducir las fricciones, son los siguientes:

- El movimiento rotacional entre eje y polea.
- El movimiento rotacional entre poleas o polea y barra.

El primer problema que se plantea de la rotación libre respecto del eje para no transmitirle ningún tipo de momento de torsión, es debido a que la transmisión se realizará mediante poleas, y además algunas deberán ser poleas locas para redirigir los momentos, estas deberán girar libremente.

Por este motivo se realizará la conexión con el eje mediante **rodamientos de agujas**. Estos rodamientos están formados por un anillo el cual contiene pequeños cilindros, llamados agujas por su tamaño, que rotan libremente y están en contacto directo con el eje, mientras que el anillo va conectado a la polea por la parte externa de éste. De este modo se consigue evitar la interferencia de movimientos entre la polea y el eje que la sujeta.

HK08-10



Ilustración 6. Rodamientos de agujas KH08-10

Estos rodamientos son adecuados para conexiones con agujeros pequeños, en otros casos con agujeros más grandes se debería usar rodamientos radiales de bolas.

Por otro lado como se ha comentado anteriormente, se debe separar los movimientos rotacionales entre poleas o poleas y barra, este problema surge del posible contacto de varios elementos con diferentes velocidades rotacionales en el mismo eje, por este motivo se debe asegurar la independencia de movimiento entre los elementos, pero a la vez pudiendo mantener su posición en el eje.

Para asegurar que los elementos no interfirieran entre sí en sus movimientos rotacionales se analizó alguna que otra solución como la de anclar los cojinetes al eje. Finalmente se ha decidido utilizar rodamientos axiales de bolas, los cuales separan dos elementos permitiendo que cada uno realice su movimiento sin interferir en el otro.

F8-16M



Ilustración 7. Cojinetes F8-16M

Elementos de fijación

Una vez se consiguen aislar los elementos de transmisión para que no exista interferencia de movimientos, se deben asegurar que sus posiciones respecto al eje no cambian, ya que podría suponer un mal funcionamiento del brazo o directamente un error fatal para éste. Las fijaciones más importantes que surgen en el proyecto, son las siguientes:

- La fijación de los elementos en los ejes para evitar su desplazamiento.
- La fijación de los ejes respecto a la estructura.

Para fijar los elementos en los ejes, evitando así su desplazamiento respecto al eje longitudinal de éste, se utilizarán anillos de fijación los cuales consiguen fijar su posición mediante el uso de tornillos a presión contra el eje. Esta solución se puede considerar óptima conociendo que con los rodamientos presentados anteriormente, se consigue una separación de los elementos, teniendo

por un lado el anillo de fijación que evitará desplazamientos, y por el otro lado el rodamiento axial de bolas el cual separará este anillo de fijación de los elementos a fijar sin tener ningún tipo de interferencia más que la de fijar una posición.



Ilustración 8. Anillos de fijación

Como se observa en esta ilustración el anillo de fijación está formado por el mismo anillo como tal y uno o dos tornillos de presión. Tal y cómo se ha explicado anteriormente, el funcionamiento de fijación es simple pero eficaz ya que viene a ser un apriete sin juego.

La segunda fijación que hemos mencionado en el listado anterior, es la del eje respecto a las barras estructurales del brazo robótico. Para realizar esta configuración, existe un mecanismo o elemento en el mercado muy útil para este caso, se trata de un anillo de fijación colocado dentro de un rodamiento de bolas y éste en un soporte el cual puede ir atornillado. A continuación se muestra una ilustración detallada de este elemento.

KFL08



Ilustración 9. Anillo de fijación KFL08

Con este elemento se fijará el eje mediante su anillo de fijación, de este modo el eje no podrá realizar ningún movimiento de translación respecto su eje longitudinal pudiendo, pero, rotar libremente gracias al rodamiento de bolas que rodea el anillo. El otro movimiento que se debe fijar es el de translación con respecto a la normal del eje, para ello se utilizará el soporte del rodamiento el cual irá atornillado a la estructura para así bloquear el movimiento del eje.

Sistema de rotación de la base

La rotación de la base es fundamental para este trabajo ya que un brazo robótico sin su principal articulación no sería demasiado funcional o limitaría su movimiento a solamente dos dimensiones.

Para ello se han analizado varias opciones como las de realizar este movimiento a través de un sistema de engranajes, o por otro lado con un sistema de correa y polea dentada como se pretende realizar con las articulaciones del brazo, pero finalmente, gracias a un elemento que viene a ser un rodamiento de bolas de gran diámetro pero de un espesor bajo, y preparado con 8 taladros, 4 en cada anillo del rodamiento, ofrece la posibilidad de atornillarlo por un lado a la base y por otro lado al empotramiento que se va a realizar. De este modo conectando un servomotor fijo con una chapa metálica al disco que va atornillado a la base, permitirá controlar la rotación de la base.

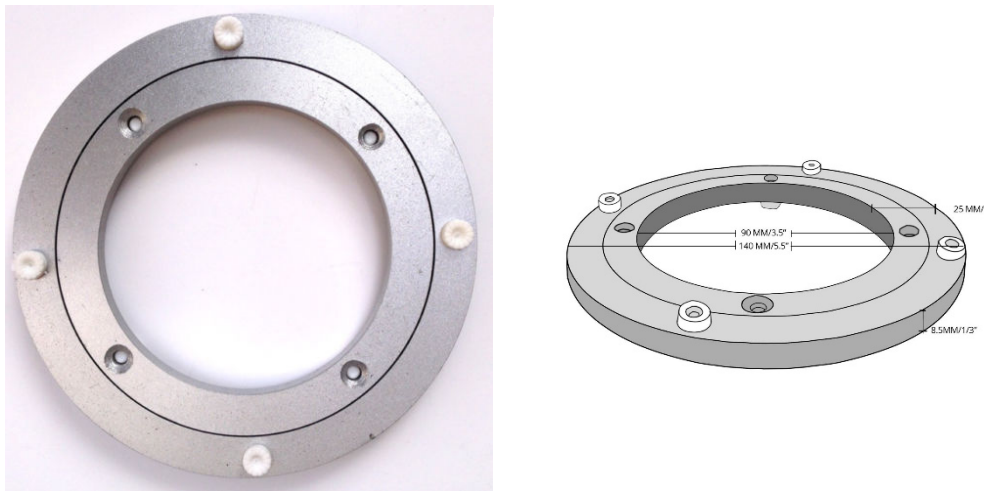


Ilustración 10. Articulación de la base del prototipo

Como se puede observar en la ilustración 10 el rodamiento ya está preparado para su instalación, por lo que solo se deberá acondicionar el actuador para que tenga un funcionamiento correcto en esta aplicación.

5.2. Diseño de las articulaciones

A continuación se explicará detalladamente el diseño del brazo robótico parte por parte y el aspecto que tomará según las decisiones que se tomen, por otro lado cabe destacar que todo diseño se basa en gran parte en un método iterativo en el cual se añaden nuevos elementos y luego se deben dimensionar los demás según la influencia de estos, es decir, cualquier modificación que se realice se debe comprobar su influencia sobre el resto del diseño y dimensionar de nuevo si es necesario. El aspecto general del brazo diseñado va estrictamente relacionado a las necesidades y/o solicitudes que se han puesto anteriormente como características técnicas, por lo tanto, éstas se deberán cumplir por completo.

Seguidamente se muestra el listado de partes del brazo que se diseñaran como módulos ya que se deben realizar los cálculos articulación a articulación:

- Pinza
- Muñeca
- Codo
- Hombro
- Cintura

5.2.1. Pinza

La pinza es uno de los elementos más interesantes del sistema robótico y el que se ha considerado inicialmente. La especificación geométrica era única y consistía en garantizar una apertura de 0.05 m.

La apertura y el cierre de la pinza lo proporcionará un servomotor. Hoy en día existen numerosos diseños de pinzas con este principio de funcionamiento y su inmensa mayoría hace uso de engranajes.

En cuanto al diseño del brazo en cuestión, debido a las sollicitaciones será necesario que la pinza disfrute de una buena amplitud de movimiento, pues debe ser capaz de coger correctamente elementos de amplitud máxima 5 cm con cierta comodidad.

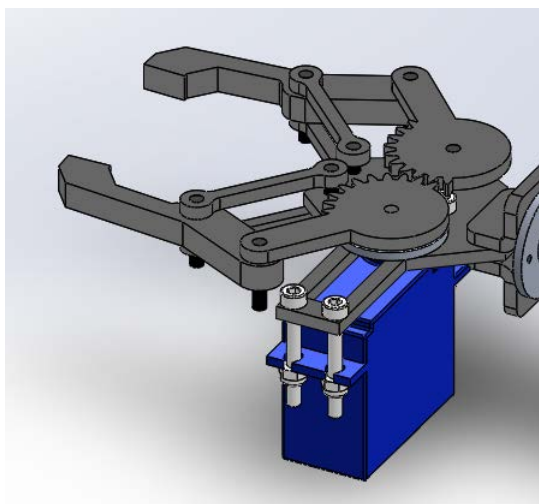


Ilustración 11. Pinza del prototipo

El servomotor tendrá acoplado a su eje de salida un engranaje, el cual está asociado a uno de los extremos de la pinza. Este engranaje, estará al mismo tiempo engranado a otro totalmente simétrico al primero conformando la pinza. Para garantizar el correcto movimiento, en cada extremo de la pinza se acopla también a dos manivelas por encima y por debajo.

El ensamblaje de cada una de las partes se realizará con tornillos, tuercas y arandelas para obviar el mínimo rozamiento existente. Con una rotación de unos 54 grados del servomotor, se garantizaría la apertura solicitada de 5 cm en las especificaciones.

Para información adicional en relación a geometrías, de ahora en adelante, **véase el Anexo A: Planos del prototipo.**

5.2.2. Muñeca

Esta parte del robot es la única que contiene dos movimientos, el primero se basa en conseguir hacer rotar la pinza para adaptarse a la geometría del objeto que tenga que coger, mientras que el segundo se trata básicamente del levantamiento de la pinza. A continuación se explicará el diseño realizado para conseguir cada uno de los movimientos especificados y los elementos mecánicos que se han utilizado en ello.

Para la rotación de la pinza se había pensado inicialmente en realizarlo con una transmisión de engranajes cónicos, la cual a la vista de cualquier ingeniero viene a ser un elemento realmente bello, pero debido a la complejidad que suponía, y gracias a las piezas que existen en el mercado orientadas directamente para el modelismo con servomotores, se decidió realizar esta rotación con una conexión directa a la salida de un servomotor tal y como se puede apreciar en la siguiente imagen.

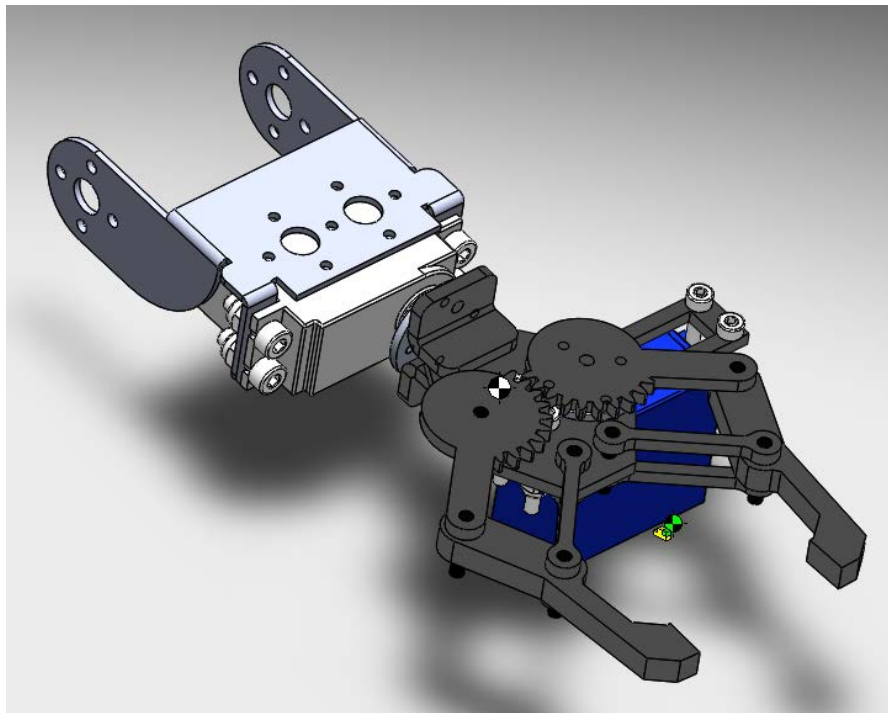


Ilustración 12. Configuración de la muñeca del prototipo

En segundo lugar para realizar el movimiento de levantar la pinza, como ya se ha comentado anteriormente, la transmisión de los pares motrices de los motores se realizará mediante correas y poleas dentadas directamente ancladas al lateral de la barra a la cual se quiere transmitir el par. Una vez decidido este diseño lo siguiente será colocar los elementos principales de transmisión los cuales constan de un eje libre, la polea de transmisión, los rodamientos de la polea, y los tornillos y tuercas para anclar la barra a la polea transmisora. Aplicando estos elementos el diseño evoluciona de la siguiente manera:

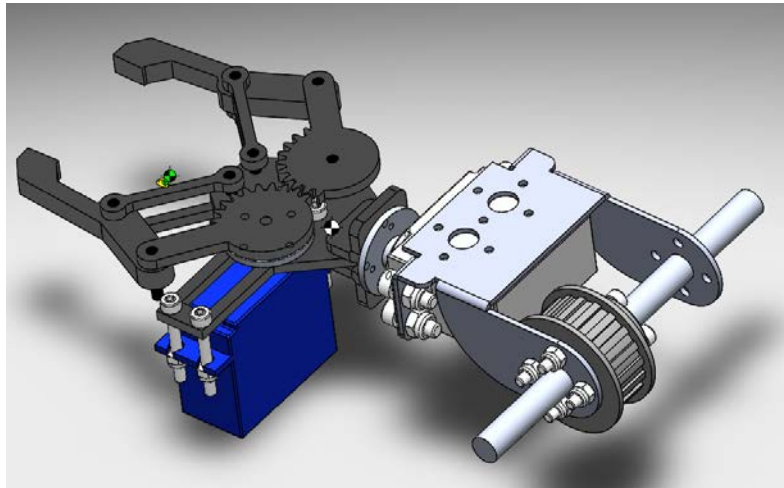


Ilustración 13. Diseño pinza prototipo

Una vez colocados los elementos principales de transmisión, se debe pensar como mantener los elementos fijos en la dirección del eje de rotación para que no se puedan desplazar en éste y a la vez aislarlos de los laterales de la estructura del brazo que irán acopladas al eje. Para mantener la posición fija en el eje, en el apartado anterior se han descrito los elementos tanto de fijación como de posicionamiento para evitar interferencias los cuales son fundamentales tanto para poder ensamblar la muñeca con el siguiente escalón del brazo robótico como para mantener ésta en su posición dentro del eje.

En primer lugar se debe colocar el anillo de fijación juntamente con el rodamiento axial, de esta manera el soporte de la muñeca no se podrá desplazar hacia la dirección X del eje ya que estos elementos lo bloquearán.

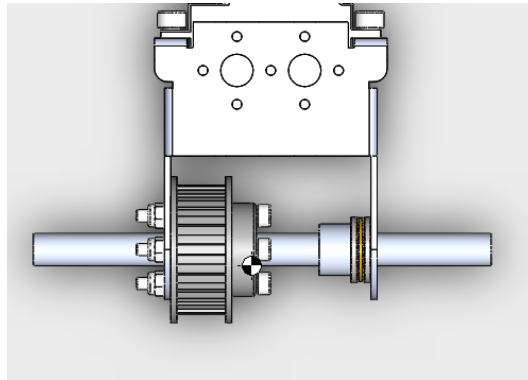


Ilustración 14. Fijación de la pinza al eje(I)

Seguidamente se situarán dos rodamientos axiales en el lado externo del soporte, estos rodamientos a la vez que dar distancia hasta el primer eslabón estructural del brazo lo aísla de interferencias cinéticas.

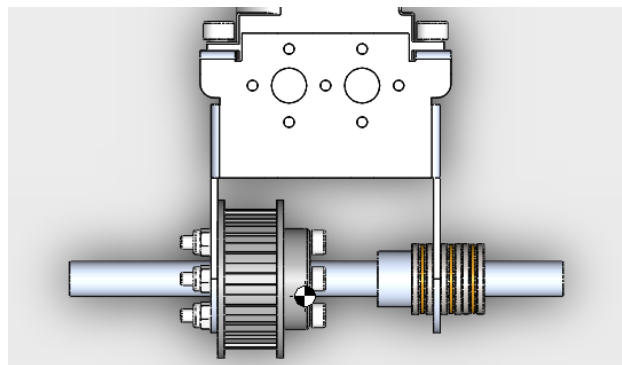


Ilustración 15. Fijación de la pinza al eje (II)

Una vez acondicionada la muñeca y sus elementos se incorporarán las barras estructurales del brazo robótico, las cuales llevan atornilladas los soportes de fijación para el eje. De este modo se tendrá en este momento la muñeca completamente cerrada y situada dentro del siguiente eslabón.

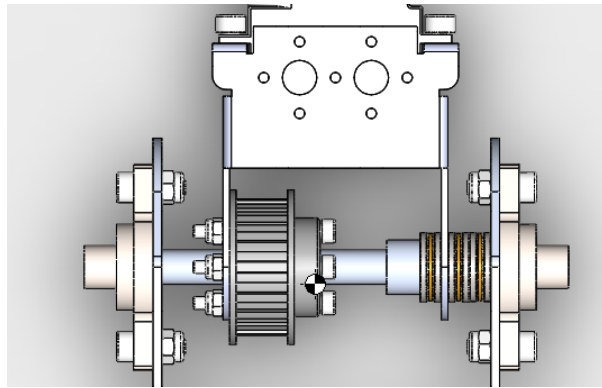


Ilustración 16. Fijación de la pinza al eje (III)

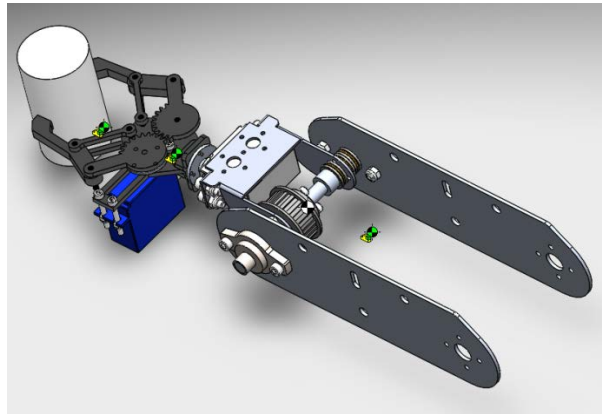


Ilustración 17. Diseño final muñeca

5.2.3. Codo

Siguiendo el patrón de diseño establecido para el brazo robótico, la articulación del codo se configurará de la misma manera que la muñeca, colocando la polea transmisora directamente acoplada a la barra sobre la que actuará para transmitir el par hacia toda la extremidad que le sigue, acondicionada mediante rodamientos de agujas para su movimiento libre respecto al eje y atornillada a la barra a la cual le transmite el par.

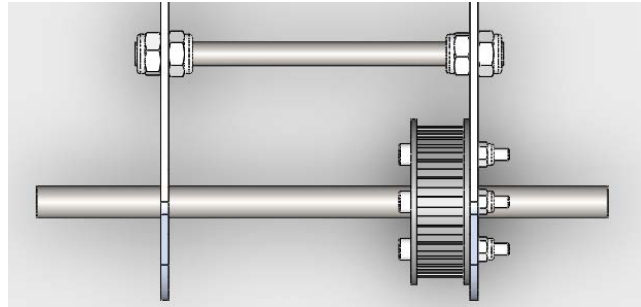


Ilustración 18. Fijación de la articulación del codo al eje(I)

Por otro lado se deberá colocar la polea transmisora que realizará un reenvío del movimiento del motor hasta la muñeca. Una vez establecidos los principales elementos de transmisión de deberán acondicionar para su correcto funcionamiento de igual manera que se realizará en la muñeca, colocando tanto anillos de fijación como rodamientos axiales en puntos estratégicos para fijar la posición y evitar movimientos indeseados.

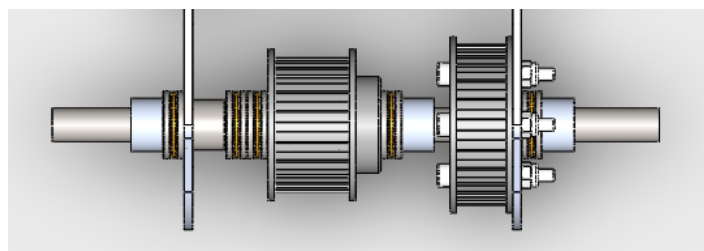


Ilustración 19. Fijación de la articulación del codo al eje(II)

En primer lugar se fija la posición de la barra a la cual se le transmite el par con un rodamiento axial en contacto con un anillo de fijación, de esta manera se evita el desplazamiento de la barra hacia ese sentido.

Seguidamente con la colocación del mismo sistema en contacto con la polea de reenvío se fija por un lado su posición, mientras que por el otro lado con dos rodamientos axiales, los cuales dan una separación necesaria y evitan la interferencia con el rodamiento radial al que va conectado la segunda barra, otorgan una fijación libre de interferencias para finalmente usar el sistema rodamiento mas anillo de fijación para asegurar la segunda barra estructural. De este modo se consigue que la segunda articulación esté bien situada y libre de desplazamientos no deseados fijando de tal manera los componentes que estos pueden trabajar correctamente y cumplir su función.

Finalmente para cerrar definitivamente la segunda articulación, se le añaden los soportes para atornillarlos a las barras que conectan el codo con el siguiente eslabón, el hombro.

5.2.4. Hombro

Una vez más siguiendo el patrón de diseño para las articulaciones que realizan el mismo movimiento de levantamiento como son la muñeca, codo y hombro, se adaptará al eje de la articulación los elementos correspondientes a la posición y fijación conjuntamente a los de transmisión o reenvío de transmisiones.

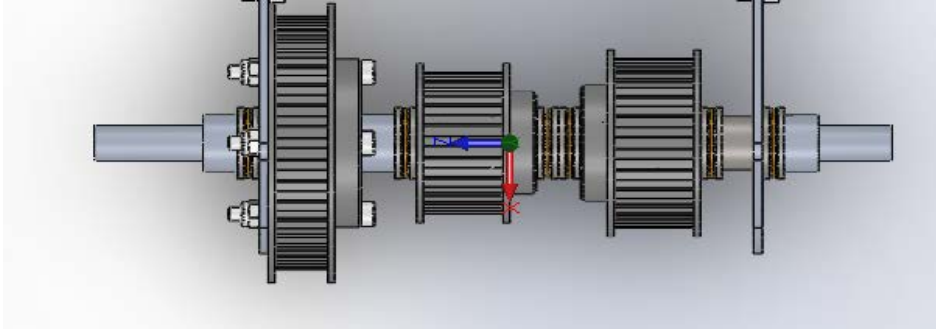


Ilustración 20. Fijación de la articulación del hombro al eje(I)

De este modo lo único que nos quedará para las transmisiones, es unirlas a los motores que darán el par de salida a transmitir y los cuales estarán colocados en la base y en posiciones estratégicas.

5.2.5. Cintura

La parte de la cintura o base es donde convergen todos los sistemas de transmisión con los pares motrices de los motores, juntamente con el principal movimiento de rotación del brazo robótico. Para cumplir todas las funciones se ha diseñado una base donde por un lado se acopla el eje del hombro y por otro lado se alojan los motores.

El encaje del eje del hombro se realiza mediante el uso de los soportes usados en todas las articulaciones los cuales integran el sistema de encaje mediante tornillos y un rodamiento de bolas para permitir su libre giro.

Los motores van situados a las distancias calculadas en el **anexo A** (tal y como se han dimensionado las transmisiones de los pares).

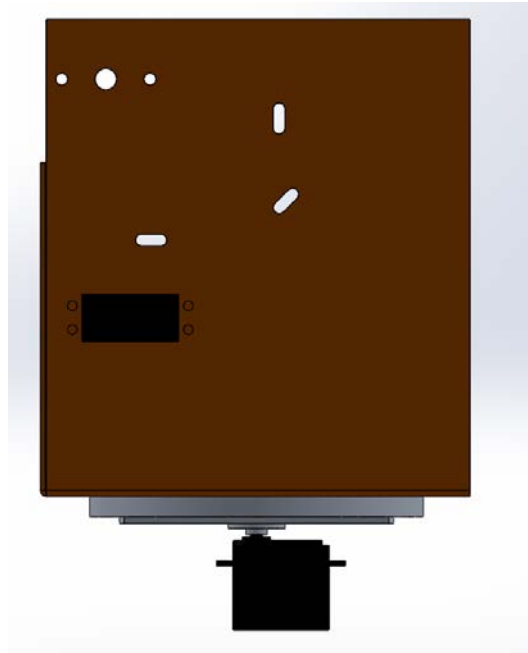


Ilustración 21. Configuración de la base (I)

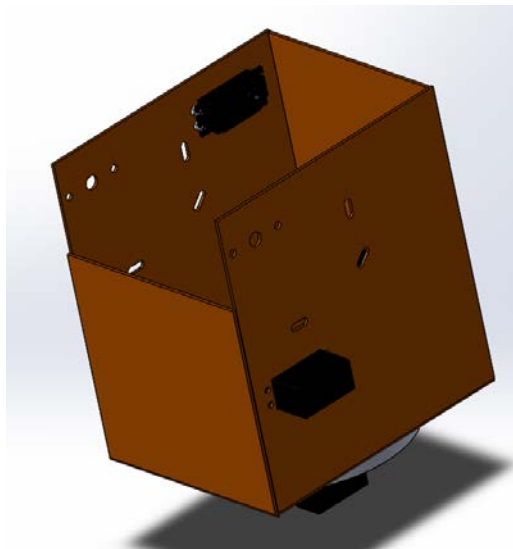


Ilustración 22. Configuración de la base (II)

Para conseguir realizar el movimiento de la cintura, como se ha comentado anteriormente se habían pensado otras soluciones en un principio, pero con el rodamiento de gran diámetro y con el acondicionamiento que otorgan los agujeros de fábrica resulta una solución más sencilla a la par que eficaz. Primero de todo se debe adaptar el sistema para acoplar un servomotor, en este caso se debe tener en cuenta que el servomotor debe estar sujeto a alguna parte para que funcione correctamente, y por otro lado se debe acoplar la salida a la base.

Para adaptar el motor al sistema se usa una lámina de 1 mm de espesor de aluminio en la cual se atornillara por un lado la salida del servomotor, y por otro lado se unirá al rodamiento interno y éste a la base del brazo robótico.

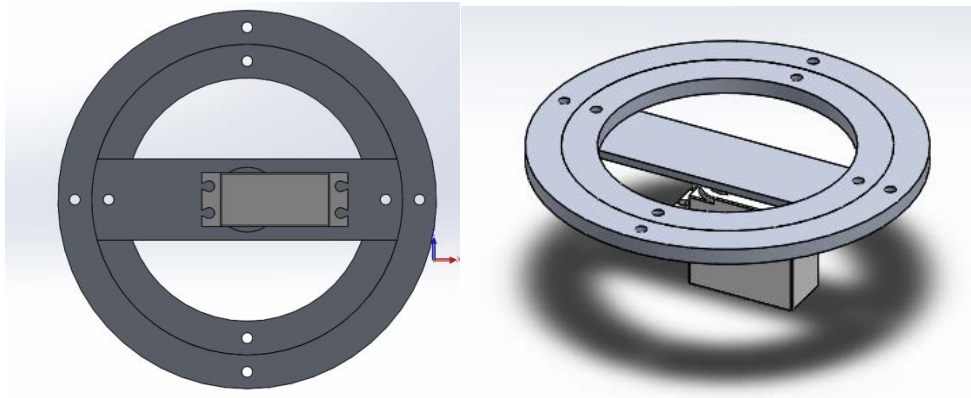


Ilustración 23. Configuración de la base (III)

Una vez unidos estos elementos se unirá a la parte externa del rodamiento una base externa de gran masa para que sirva de empotramiento para todos los esfuerzos de la base, mientras que por otro lado se fijará a ésta base externa dos soportes que mantengan el servomotor atado por ambos lados para evitar el movimiento de rotación y así asegurar su correcto funcionamiento.

5.2.6. Barras estructurales

Estos elementos son muy importantes ya que son los que conectan las articulaciones entre si hasta llegar a la base. A la vez que ser resistentes deben permitir la conexión de todos los elementos y asegurar su correcto funcionamiento, por eso se deben tener en cuenta características como el material, la forma o su espesor para conseguir una estructura sencilla pero funcional. El primer paso de diseño de la estructura es sencillo ya que se conoce que debe conectar dos ejes y ser lo más similar a una línea entre ambos, seguidamente partiendo de los conocimientos sobre física y resistencia de los materiales se sabe qué dirección tendrán los esfuerzos y qué secciones serán las que sufran más, y es por ello que se decide usar chapa metálica de aluminio colocada verticalmente con su superficie perpendicular a los ejes. Una vez se tiene la forma inicial de las barras, se debe pensar en cómo se acondicionarán para su conexión tanto con los ejes como con las transmisiones y como asegurar su correcto funcionamiento.

En primer lugar se pretende cumplir la especificación técnica del brazo robótico, en la cual nos dice una distancia mínima de alcance y ésta dependerá de la distancia que tengan los eslabones del brazo robótico. Por lo tanto se debe decidir la longitud de los brazos la cual será de unos 15 cm por cada articulación, lo que implica directamente que los agujeros a realizar para las conexiones con los ejes deben estar a una distancia de 15 cm entre centros.

Una vez realizado un primer croquis de las barras, se debe tener en cuenta como se ha mencionado anteriormente que se deberá estabilizar la estructura con barras estabilizadoras para evitar las

deformaciones no deseadas debido a su asimetría. Por lo tanto se acondiciona la estructura realizando agujeros en puntos estratégicos para estabilizarla al máximo.

Finalmente al pretender realizar la transmisión mediante un sistema de poleas y correas dentadas, es muy importante tener en cuenta que es un sistema elástico, el cual se debe tensar según el tiempo, por lo tanto es imprescindible realizar un sistema de tensado para las correas, por este motivo se realizan unos colisos en la estructura para pasar barras ajustables con poleas tensoras para cumplir este objetivo.

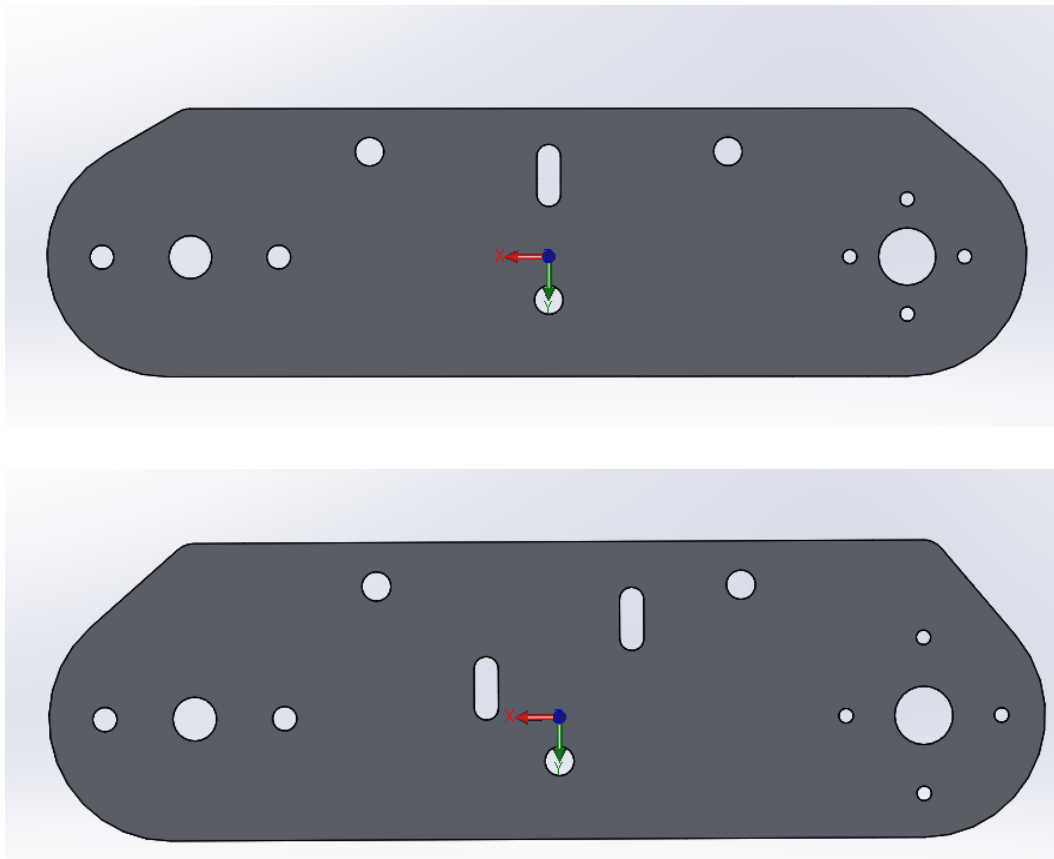


Ilustración 24. Barras estructurales

5.3. Evaluación del conjunto

5.3.1. Cálculos estáticos sobre articulaciones

Uno de los cálculos más sencillos y a la vez más trascendentales que surge es la definición del par nominal necesario en cada servomotor para cumplir las solicitaciones de carga máxima en cualquier configuración del brazo. En este caso, se propondrá la configuración más crítica para fijar las solicitaciones del diseño global.

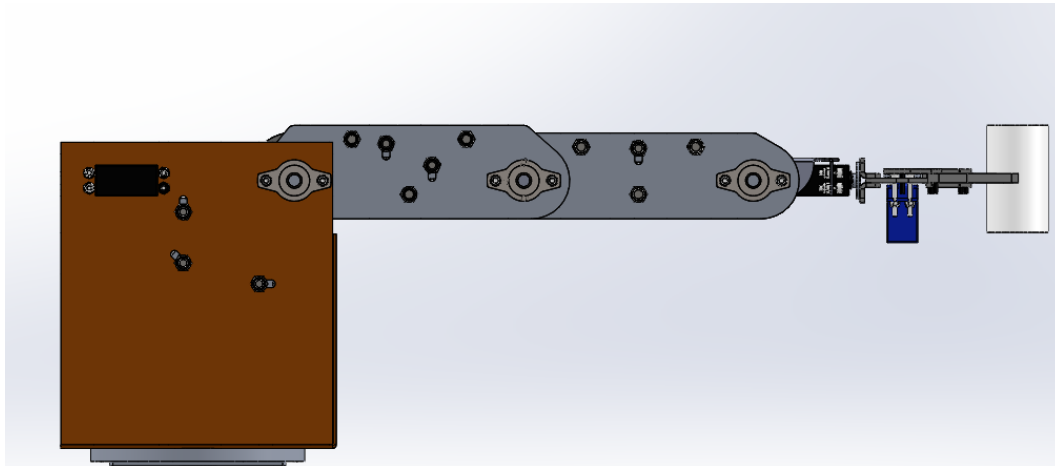


Ilustración 25. Configuración crítica

Para cada una de las articulaciones del brazo se ha realizado el estudio estático y dinámico para obtener el momento de torsión a superar para poder mover y aguantar estáticamente el brazo robótico.

En ambos casos se ha hecho soporte del Software SolidWorks y su función Evaluate la cual proporciona la información para realizar los cálculos.

Considerando como unidades básicas de par o momento como N·m, evidentemente como más lejos se aplique la fuerza, mayor será el par máximo a soportar por el motor. La configuración más crítica es aquella en la que el brazo está completamente estirado horizontalmente, tal y como se observa en la ilustración anterior.

Los cálculos se empiezan por la parte más alejada de la base ya que su configuración será la que marcará los siguientes pasos.

Para cada una de las articulaciones se debe realizar el análisis estático y dinámico para obtener la sollicitación de diseño.

5.3.2. Análisis estático

En el cálculo estático se desea encontrar el par solicitado para mantener el sistema en equilibrio sin que haya movimiento. Para realizar este análisis enfocado al par solicitado, se debe solucionar la ecuación en la que el equilibrio se cumple con el par del motor.

Para este cálculo se deben tener cuenta la masa de los sólidos y las distancias de sus centros de gravedad al centro de rotación donde se aplicará el par del motor. Por otro lado se puede simplificar el sistema cogiendo la masa total del conjunto de sólidos, y su centro de gravedad global, de ese modo se simplifica la ecuación. Estos parámetros se obtienen gracias al Software de diseño, ya que una vez realizada la modelización, esta herramienta nos devuelve los datos necesarios para realizar los cálculos.

Según la ley de Newton, para tener un sistema en equilibrio se deben cumplir varias condiciones:

$$\sum F = 0 \quad [1]$$

$$\sum M = 0 \quad [2]$$

$$\text{Peso Barra/Sistema } X = P_X$$

$$\text{Reacción Articulación } X = R_X$$

$$\text{Longitud Centro de gravedad Barra/sistema – Articulación} = L_{cgX}$$

Las reacciones de los esfuerzos cortantes se irán transmitiendo hasta la base la cual realizará la reacción total para anular este esfuerzo cortante y que exista el equilibrio estático.

$$P_X - R_X = 0 \quad [3]$$

Por otro lado los momentos o pares se calculan gracias a los parámetros obtenidos de la simplificación que se ha realizado anteriormente, con el centro de gravedad del sistema y la distancia de este a la articulación, de este modo se obtiene el par mínimo que debe tener el motor o la transmisión.

$$P_X \cdot L_{cgX} = M_{E_{R_X}} \quad [4]$$

El análisis estático se realizará según se ha especificado anteriormente, cabe destacar que para este análisis se omitirán los elementos cuyo centro de gravedad este ubicado en cualquier punto de la vertical del eje de rotación, por lo tanto el sistema para el análisis estático quedara tal y como se muestra en las figuras correspondientes a las articulaciones.

Pinza:

En este punto se calculará la fuerza necesaria para sujetar el objeto. Para esto se debe analizar la forma de la pinza y saber dónde se aplica el par del motor.

Considerando que la localización del centro instantáneo de rotación (CIR) se localiza casi al infinito, el movimiento del extremo de la pinza se puede considerar que es un movimiento de traslación. El servomotor aportará al sistema un par Γ , que, debido a la simetría del sistema, se considerará llanamente que se repartirá a partes iguales en cada extremo de la pinza.

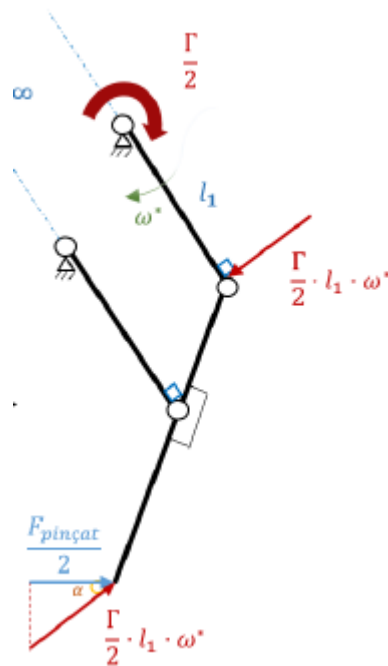


Ilustración 26. Análisis estático pinza (I)

La resultante horizontal que la pinza comunicará al cuerpo a cargar, dependerá en gran medida de su apertura, por lo que habrá que aplicar la trigonometría correspondiente como se ilustra en la ilustración 26.

Mediante el planteamiento de las dos situaciones extremas, la fuerza mínima de 2 N lleva a la obtención del par mínimo necesario del servomotor de la pinza, tal y como se representa a continuación:

$$F_{\text{Pinzado}} = \frac{M}{l_1} \cdot \cos(\alpha), \text{ donde } l_1 = 0.318 \text{ m [5]}$$

$$F_{\text{Pinzado mín.}}(65^\circ) = 2 \text{ N} \rightarrow M_{\text{mín}} = 0.15 \text{ N} \cdot \text{m}$$

$$F_{\text{Pinzado máx.}} M_{\text{mín}}(11^\circ) = 4.6 \text{ N}$$

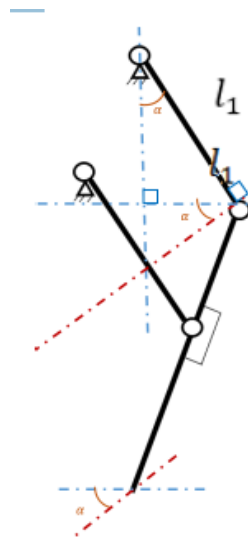


Ilustración 27. Análisis estático pinza (II)

Ya por último, añadir que habrá que adherirse a las puntas de la articulación una fina capa de goma para garantizar el correcto apoyo de la carga y evitar deslizamientos. El coeficiente de fricción mínimo viene definido a partir de los parámetros anteriores y la carga máxima:

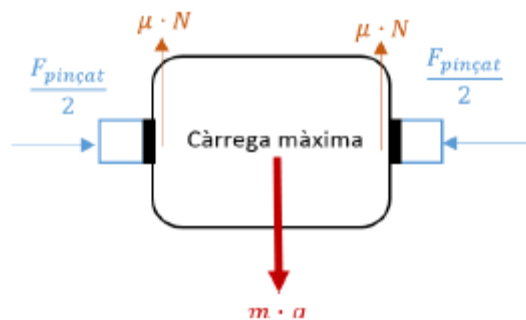


Ilustración 28. Análisis estático pinza (III)

En este caso también se realizará el cálculo para el par estático del giro de la muñeca, esto es debido a la excentricidad existente tanto en el diseño de ésta, cómo en la posibilidad de coger el objeto por uno de sus extremos, lo cual generaría una excentricidad añadida que desplazaría el centro de gravedad del conjunto.

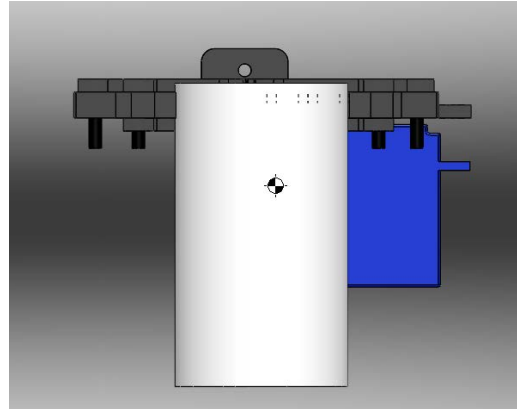


Ilustración 29. Análisis estático pinza (IV)

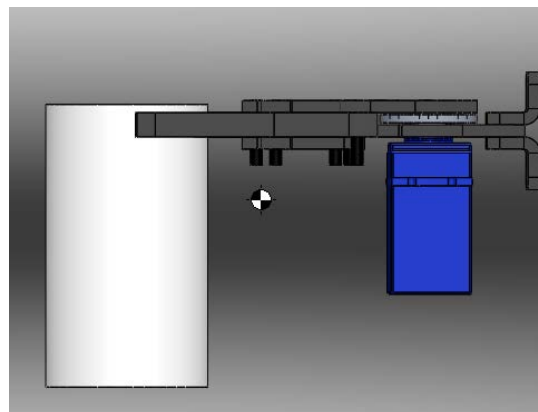


Ilustración 30. Análisis estático pinza (V)

Como se puede observar en las anteriores ilustraciones, el centro de gravedad del conjunto se ha desplazado, por lo tanto, en el caso más crítico de este, es decir, cuando la pinza esté en paralelo con el plano vertical, el peso del conjunto forzará un par estático para mantener la pinza en esta posición.

$$P_{Pinza} = 210 \text{ gr} \cdot \left(\frac{1 \text{ kg}}{1000 \text{ gr}} \right) \cdot 10 \frac{\text{m}}{\text{s}^2} = 2.1 \text{ N}$$

$$L_{Cg_x} = 1.71 \text{ cm} \cdot \frac{1 \text{ m}}{100 \text{ cm}} = 0.0171 \text{ m}$$

$$M_{EstáticoMuñeca} = P_{Muñeca} \cdot L_{Cg_x}$$

$$M_{EstáticoMuñeca} = 2.1N \cdot 0.0171 m = \mathbf{0.0359 N \cdot m}$$

Muñeca:

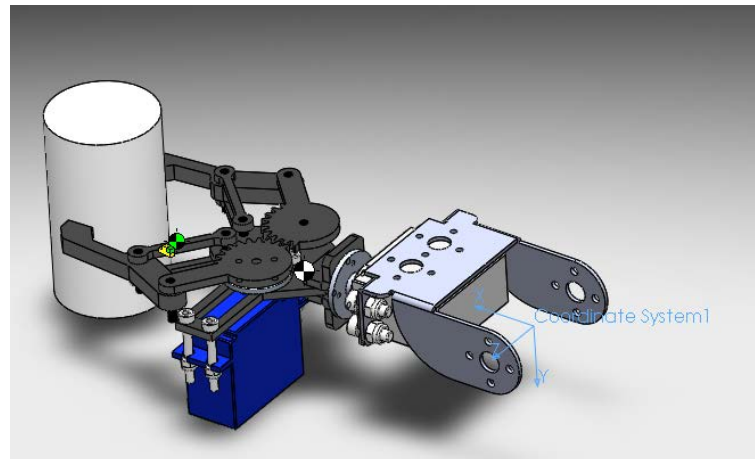


Ilustración 31. Análisis estático muñeca

Los datos obtenidos a partir de la herramienta "Evaluate" de SolidWorks nos permite calcular de una manera sencilla y simplificada el par estático mínimo que debemos aplicar a la articulación.

$$P_{Muñeca} = 320 gr \cdot \left(\frac{1kg}{1000gr} \right) \cdot 10 \frac{m}{s^2} = 3.2 N$$

$$L_{CgX} = 10.84 cm \cdot \frac{1m}{100cm} = 0.1084 m$$

$$M_{EstáticoMuñeca} = P_{Muñeca} \cdot L_{CgX}$$

$$M_{EstáticoMuñeca} = 3.2 N \cdot 0.1084 m = \mathbf{0.3468 N \cdot m}$$

Codo:

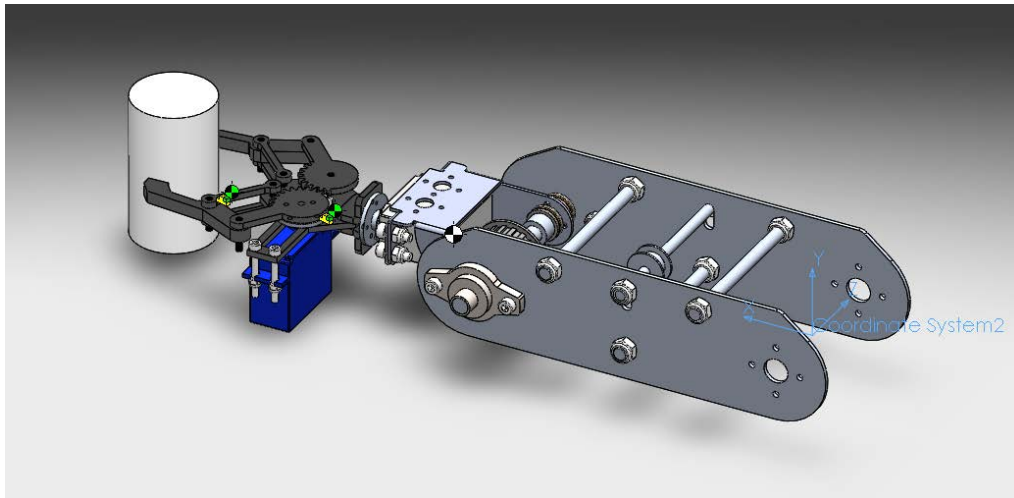


Ilustración 32. Análisis estático codo

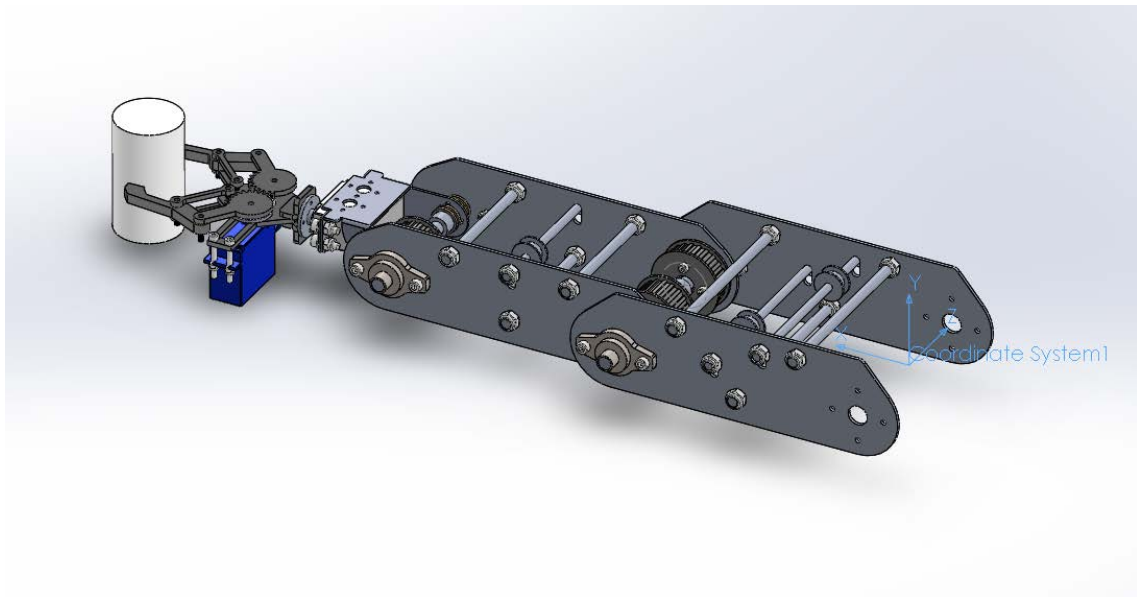
Los datos obtenidos a partir de la herramienta “Evaluate” de *SolidWorks* nos permite calcular de una manera sencilla y simplificada el par estático mínimo que debemos aplicar a la articulación.

$$P_{Codo} = 680 \text{ gr} \cdot \left(\frac{1 \text{ kg}}{1000 \text{ gr}} \right) \cdot 10 \frac{\text{m}}{\text{s}^2} = 6.8 \text{ N}$$

$$L_{Cg_x} = 18.03 \text{ cm} \cdot \frac{1 \text{ m}}{100 \text{ cm}} = 0.1803 \text{ m}$$

$$\mathbf{M}_{EstáticoCodo} = P_{Codo} \cdot L_{Cg_x}$$

$$\mathbf{M}_{EstáticoCodo} = 6.8 \text{ N} \cdot 0.1803 \text{ m} = \mathbf{1.2261 \text{ N} \cdot \text{m}}$$

Hombro:*Ilustración 33. Análisis estático hombro*

Los datos obtenidos a partir de la herramienta "Evaluate" de SolidWorks nos permite calcular de una manera sencilla y simplificada el par estático mínimo que debemos aplicar a la articulación.

$$P_{Hombro} = 1150 \text{ gr} \cdot \left(\frac{1\text{kg}}{1000\text{gr}} \right) \cdot 10 \frac{\text{m}}{\text{s}^2} = 11.5 \text{ N}$$

$$L_{Cg_x} = 24.15 \text{ cm} \cdot \frac{1\text{m}}{100\text{cm}} = 0.2415 \text{ m}$$

$$M_{EstáticoHombro} = P_{Hombro} \cdot L_{Cg_x}$$

$$M_{EstáticoHombro} = 11.5 \text{ N} \cdot 0.2415 \text{ m} = 2.7772 \text{ N} \cdot \text{m}$$

Tabla 9. Par de sollicitación estática

Articulación	Par de sollicitación estática (N·m)
Cierre Pinza	0.15
Giro Pinza	0.04
Muñeca	0.35
Codo	1.23
Hombro	2.78

5.3.3. Análisis dinámico

El cálculo dinámico nos permite obtener los momentos necesarios para el sistema con la configuración deseada de velocidades y aceleraciones. En este estudio se debe tener en cuenta los cálculos del análisis estático ya que se debe superar el momento calculado más el momento a realizar dinámicamente. Para el cálculo del par dinámico se parte de los datos del Software de diseño, en el cual se han adaptado unos sistemas de coordenadas en las articulaciones para así, obtener las matrices de inercia en estos sistemas de referencia. Estas matrices se calculan siguiendo el teorema de Steiner de los ejes paralelos, de esta manera si el diseño está realizado correctamente, los cálculos se aproximarán de una manera muy considerable a la realidad siempre teniendo en cuenta que se trata de una modelización aproximada.

$$I = \begin{pmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{pmatrix} [6]$$

Una vez obtenida la matriz en el sistema de referencia deseado, se calcula el momento a superar con la aceleración configurada para la articulación correspondiente.

$$M_D = I_{zz} \cdot a_x [7]$$

El análisis dinámico se realizará según se ha especificado anteriormente. Los parámetros de aceleración angular y velocidad angular son los establecidos en las especificaciones técnicas a cumplir para cada una de las articulaciones, de esta manera se establecen unos parámetros de cálculo. En este caso debemos tener en cuenta todos los elementos que intervendrán en este movimiento, es decir, las poleas locas colocadas en las articulaciones anteriores también deberán ser consideradas ya que forman parte de este movimiento.

En conclusión el par mínimo necesario por el motor o transmisión en el punto en el cual realicemos el análisis será:

$$M_{Motriz} = M_{Estático} + M_{Dinámico} [8]$$

Por otro lado cabe destacar que se han establecido unos perfiles de velocidad para los movimientos de los cuales se obtiene la velocidad angular necesaria en los cálculos siguientes y la aceleración angular necesaria para el análisis dinámico.

Pinza

Tal y como se ha comentado en el apartado anterior, la excentricidad provocada por la colocación del objeto y por la forma asimétrica de la pinza por la colocación del servomotor, son la

configuración más crítica de la pinza, por lo tanto se realizara el análisis dinámico en la misma situación.

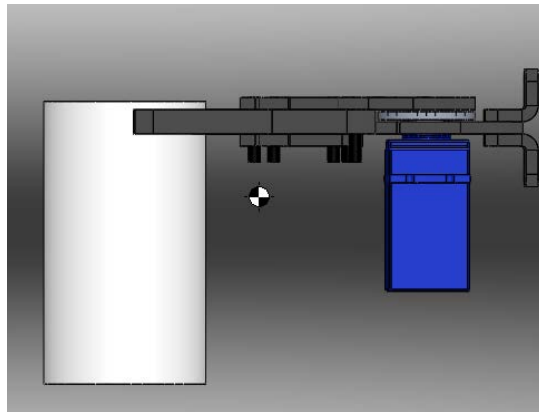


Ilustración 34. Análisis dinámico pinza

Los datos obtenidos a partir de la herramienta “Evaluate” de SolidWorks nos permite calcular de una manera sencilla y simplificada el par dinámico mínimo que debemos aplicar a la articulación.

$$Inercia\ del\ conjunto = 2216.95g \cdot cm^2$$

$$Aceleración_{Giro\ Pinza} = 64.83\ rad/s^2$$

Una vez definidos todos los parámetros necesarios para el análisis dinámico del sistema procedemos a realizar el cálculo:

$$T = \sum I \cdot a \quad [9]$$

$$T = (2216.95)(g \cdot cm^2) * 64.83 \left(\frac{rad}{s^2} \right) \cdot \left(\frac{1kg}{1000g} \right) \cdot \left(\frac{1m^2}{10000cm^2} \right)$$

$$T = 0.0143\ N \cdot m$$

Muñeca:

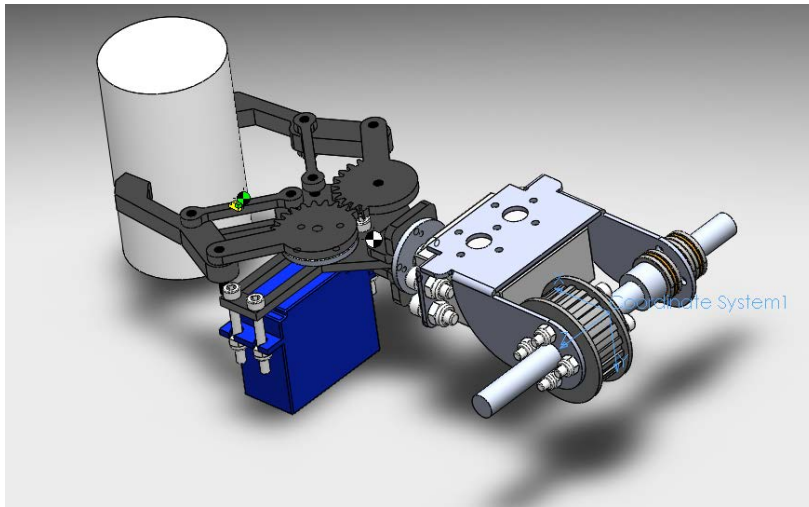


Ilustración 35. Análisis dinámico muñeca

Los datos obtenidos a partir de la herramienta “Evaluate” de SolidWorks nos permite calcular de una manera sencilla y simplificada el par dinámico mínimo que debemos aplicar a la articulación.

Todas las inercias son en el eje Z por lo tanto cogemos el valor Izz de la matriz de inercias para cada uno de los elementos.

$$\begin{aligned} \text{Inercia polea motriz} &= 47.89 \text{ g} \cdot \text{cm}^2 \\ \text{Inercia Muñeca} &= 45591.48 \text{ g} \cdot \text{cm}^2 \\ \text{Inercia poleas transmisoras} &= \frac{61.36 \text{ g} \cdot \text{cm}^2}{u} \end{aligned}$$

Como las relaciones son 1:1 según el diseño, en todos los elementos la aceleración angular será la misma, por lo tanto definiremos una única aceleración de diseño:

$$\text{Aceleración}_{\text{Muñeca}} = 20.94 \text{ rad/s}^2$$

Una vez definidos todos los parámetros necesarios para el análisis dinámico del sistema procedemos a realizar el cálculo:

$$\begin{aligned} T &= \sum I \cdot a \\ T &= (47.89 + 45591.48 + 61.36 \cdot 2)(\text{g} \cdot \text{cm}^2) \cdot 20.95 \left(\frac{\text{rad}}{\text{s}^2}\right) \cdot \left(\frac{1\text{kg}}{1000\text{g}}\right) \cdot \left(\frac{1\text{m}^2}{10000\text{cm}^2}\right) \\ T &= 4.57621 \cdot 10^{-3} \text{kg} \cdot \text{m}^2 \cdot 20.95 \frac{\text{rad}}{\text{s}^2} = \mathbf{0.0958 \text{ N} \cdot \text{m}} \end{aligned}$$

Codo:

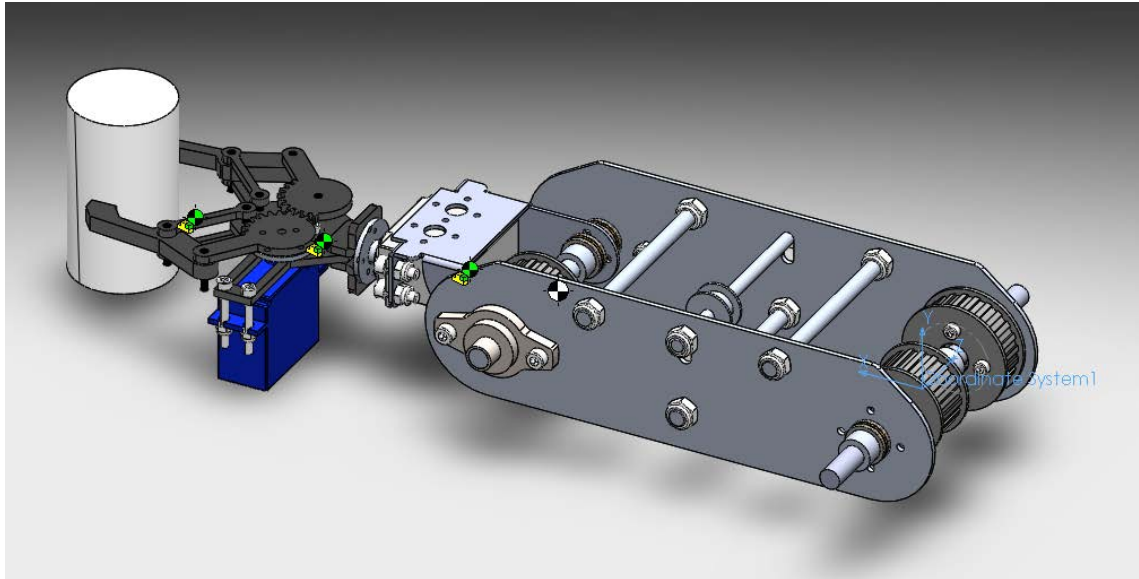


Ilustración 36. Análisis dinámico codo

Los datos obtenidos a partir de la herramienta "Evaluate" de SolidWorks nos permite calcular de una manera sencilla y simplificada el par dinámico mínimo que debemos aplicar a la articulación.

Todas las inercias son en el eje z por lo tanto cogeremos el valor Izz de la matriz de inercias para cada uno de los elementos.

$$\text{Inercia polea motriz} = 47.89 \text{ g} \cdot \text{cm}^2$$

$$\text{Inercia Muñeca} = 268584.51 \text{ g} \cdot \text{cm}^2$$

$$\text{Inercia poleas transmisoras} = 129.3 \text{ g} \cdot \text{cm}^2$$

$$\text{Relación polea motriz} - \text{conducida} = \frac{24}{20} = 1.2$$

$$\text{Relación polea conducida} - \text{salida} = \frac{24}{24} = 1$$

$$\text{Aceleración definida} = 15.71 \text{ rad/s}^2$$

$$\text{Aceleración motor} = 15.7 \cdot 1.2 = 18.84 \text{ rad/s}^2$$

Una vez definidos todos los parámetros necesarios para el análisis dinámico del sistema procedemos a realizar el cálculo:

$$T = \sum I \cdot a$$

$$T = \left((47.89)(g \cdot cm^2) * 18.84 \left(\frac{rad}{s^2} \right) + (268584.51 + 129.3) \cdot 15.71 \left(\frac{rad}{s^2} \right) \right) \cdot \left(\frac{1kg}{1000g} \right) \cdot \left(\frac{1m^2}{10000cm^2} \right)$$

$$T = (902.25 + 4221493.95) \cdot \frac{1}{10000000} = \mathbf{0.423 N \cdot m}$$

Hombro:

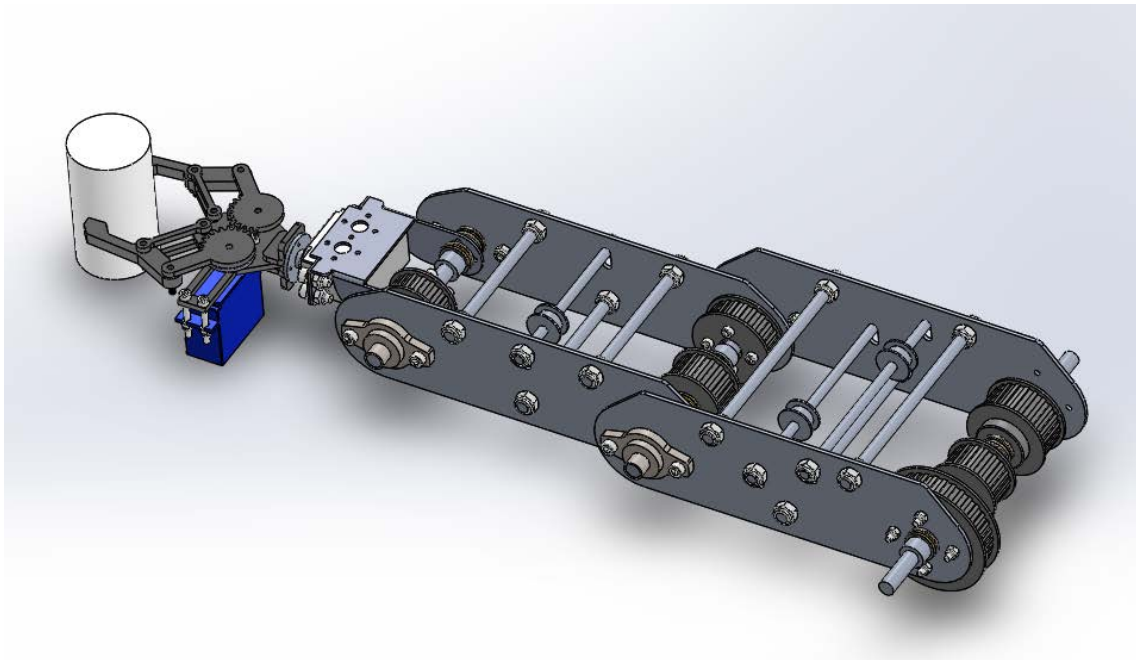


Ilustración 37. Análisis dinámico hombro

Los datos obtenidos a partir de la herramienta "Evaluate" de SolidWorks nos permite calcular de una manera sencilla y simplificada el par dinámico mínimo que debemos aplicar a la articulación.

Todas las inercias son en el eje z por lo tanto cogeremos el valor Izz de la matriz de inercias para cada uno de los elementos.

$$Inercia\ polea\ motriz = 47.89\ g \cdot cm^2$$

$$Inercia\ Hombro = 846042.58\ g \cdot cm^2$$

$$\text{Relación polea Motriz – salida} = \frac{36}{20} = 1.8$$

$$\text{Aceleración definida} = 10.47 \text{ rad/s}^2$$

$$\text{Aceleración motor} = 10.47 \cdot 1.8 = 18.84 \text{ rad/s}^2$$

Una vez definidos todos los parámetros necesarios para el análisis dinámico del sistema procedemos a realizar el cálculo:

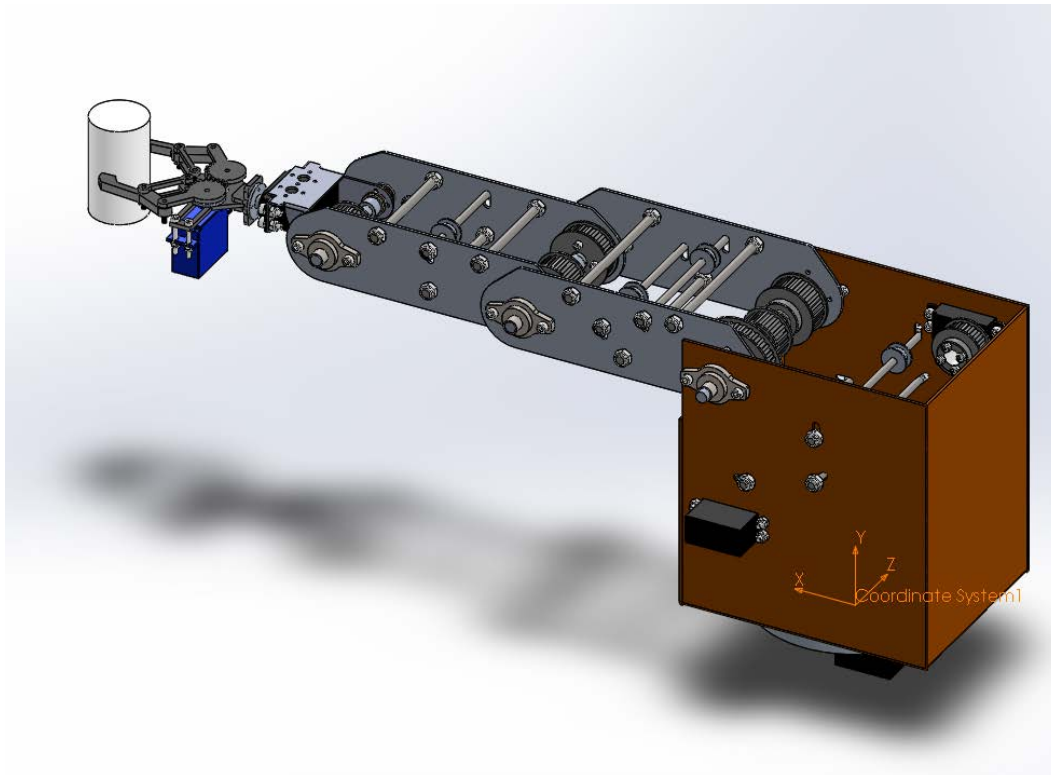
$$T = \sum I \cdot a$$

$$T = \left((47.89)(g \cdot cm^2) * 18.84 \left(\frac{rad}{s^2} \right) + (846042.58) \cdot 10.47 \left(\frac{rad}{s^2} \right) \right) \cdot \left(\frac{1kg}{1000g} \right) \cdot \left(\frac{1m^2}{10000cm^2} \right)$$

$$T = (902.25 + 8858065.81) \cdot \frac{1}{10000000} = \mathbf{0.886 \text{ N} \cdot m}$$

Base

Por último se deberá considerar el giro de la base. Para realizar el análisis de esta articulación, únicamente se realizará este análisis y no el estático debido a que no recibe ningún esfuerzo estático, el motor situado en esta articulación solamente deberá afrontar el movimiento de giro de la base y por lo tanto será un esfuerzo dinámico.



Il·lustració 38. Anàlisi dinàmico base

Siguiendo los pasos de análisis dinámicos anteriores se obtendrá el par necesario para el movimiento de rotación de la base.

$$T = \sum I \cdot a$$

$$I_{zz} = 1511412.52 \text{ g} \cdot \text{cm}^2$$

$$a = 10.47 \frac{\text{rad}}{\text{s}^2}$$

$$T = \left(1511412.52(\text{g} \cdot \text{cm}^2) * 10.47 \left(\frac{\text{rad}}{\text{s}^2} \right) \right) * \left(\frac{1\text{kg}}{1000\text{g}} \right) * \left(\frac{1\text{m}^2}{10000\text{cm}^2} \right) = 1.58 \text{ N} \cdot \text{m}$$

Tabla 10. Par de sollicitación dinàmica

Articulación	Par de sollicitación dinàmica (N·m)
Base	1.58
Giro Muñeca	0.015
Muñeca	0.096
Codo	0.423
Hombro	0.886

5.3.4. Dimensionado de los pares mínimos de los motores o transmisiones.

Con los cálculos obtenidos de los análisis anteriores se dimensionaran las fuerzas motrices mínimas para nuestro sistema. Como se ha explicado en el apartado del análisis dinámico, a este esfuerzo se le debe sumar el análisis estático ya que es un esfuerzo a superar para salir del equilibrio y entrar en dinámica. Por lo tanto es fácil llegar a la conclusión que en análisis dinámico siempre será superior al estático.

En conclusión el par mínimo necesario por el motor o transmisión en el punto en el cual realicemos el análisis será:

$$M_{Motriz} = M_{Estático} + M_{Dinámico} [9]$$

Tabla 11. Par de diseño

Articulación	Par de sollicitación estática (N·m)	Par de sollicitación dinámica (N·m)	Par de diseño
Base	0	1.58	1.58
Cierre Pinza	0.15	0	0.15
Giro Pinza	0.04	0.015	0.055
Muñeca	0.347	0.096	0.443
Codo	1.23	0.423	1.653
Hombro	2.78	0.886	3.667

Finalmente a este par de diseño se le aplicara un factor de seguridad del 20% o 1.2 para dar margen a la sollicitación.

Tabla 12. Par de diseño con factor de seguridad

Articulación	Par de diseño (N·m)	Par de modelización (N·m)
Base	1.58	1.896
Cierre Pinza	0.15	0.18
Giro Pinza	0.055	0.066
Muñeca	0.347	0.532
Codo	1.23	1.984
Hombro	3.667	4.399



6. Cinemática del brazo articulado

La cinemática del robot es el estudio de su movimiento respecto a un sistema de referencia. Normalmente este sistema de referencia, en el caso de los brazos articulados, es el de su base. Aunque también puede ser el de cualquier articulación o incluso el de la pinza.

Entonces, se tiene dos tipos de modelos de cinemática: el cinemático directo y el inverso.

La resolución del modelo cinemático directo permitirá determinar la posición y orientación del extremo del brazo respecto a un sistema de coordenadas de referencia conociendo los ángulos de las articulaciones y las características geométricas del robot.

Por otro lado, la resolución del modelo cinemático inverso, permitirá conocer la configuración que debe adoptar cada una de las articulaciones del robot para conseguir que su extremo tenga una posición e orientación conocidas.

Para que el brazo articulado pueda desplazar su extremo a través de uno de los tres ejes de coordenadas será necesario tener una función en el programa de Arduino que indique la posición en que se encuentra el robot (modelo directo) y a la vez otra función que permita calcular las posiciones que deben adoptar cada una de las articulaciones para conseguir el desplazamiento a través de un eje. Será necesaria una combinación de ambas funciones.

Para proceder con los cálculos del modelo cinemático, en la siguiente ilustración se muestra la geometría de los principales componentes del brazo que serán datos imprescindibles para su resolución.

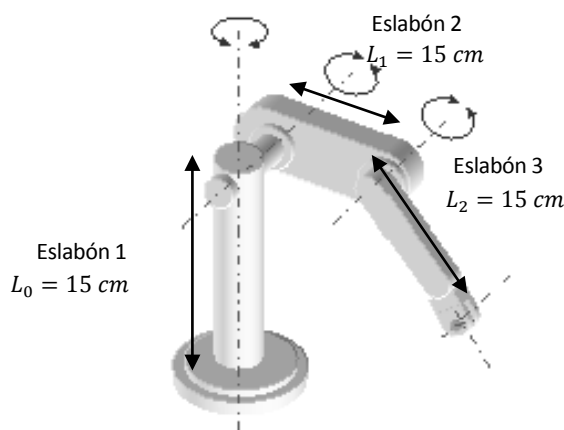


Ilustración 39. Geometría del brazo robot para cálculos de cinemática

6.1. Cálculo del modelo cinemático directo

La resolución del modelo cinemático directo del brazo articulado se puede conseguir por medio de las matrices de transformación homogénea o mediante relaciones geométricas.

A continuación se presentan ambos cálculos.

Matrices de transformación homogénea

Para éste modo de resolución se debe utilizar la representación de Denavit-Hartenberg [Miranda Colorado Roger, 2016, Cinemática y dinámica de robots manipuladores] estudiada en la asignatura de Robótica Industrial. Ésta representación se basa en el seguimiento de las siguientes pautas de procedimiento.

1. Numeración de los eslabones. Se identifica con 0 la base y a partir de aquí se enumera cada eslabón.
2. Numeración de las articulaciones. Se identifica con un 1 el primer grado de libertad y así sucesivamente.
3. Localización de los ejes de cada articulación. En el caso de las articulaciones del brazo articulado todas son de carácter rotacional y por lo tanto será el eje de giro.
4. Identificación de los ejes Z de cada articulación.
5. Identificación del sistema de coordenadas de referencia.
6. Identificación de los ejes X de cada articulación.
7. Identificación de los ejes y de cada articulación.
8. Identificación del sistema de coordenadas del extremo del robot.
9. Ángulo θ : es el ángulo entre X_{i-1} y X_i girando alrededor de Z_i .
10. Distancia d : d_i es la distancia des de el sistema XYZ_{i-1} hasta la intersección de las normales común de Z_{i-1} hacia Z_i a lo largo de Z_{i-1} .
11. Distancia a : a_i es la longitud de la normal común.
12. Ángulo α : α_i es el ángulo que hay que rotar de Z_{i-1} para llegar a Z_i rotando alrededor de X_i .

De este modo, se consiguen las matrices de transformación para cada eslabón:

$$\begin{pmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Por último se obtiene la transformación total que relaciona la base del robot con su extremo, siendo la multiplicación de las matrices de cada uno de su eslabón.

$$T = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \text{ [10]}$$

Tabla de valores para las articulaciones:

Tabla 13. Valores de la representación de Denavit-Hartenberg

	θ	d	a	α
<i>Cintura</i>	θ_1	0	0	90
<i>Hombro</i>	θ_2	0	L_1	0
<i>Codo</i>	θ_3	0	L_2	0

Matrices:

$${}^0A_1 = \begin{pmatrix} \cos \theta_1 & -\cos 90 \sin \theta_1 & \sin 90 \sin \theta_1 & 0 \\ \sin \theta_1 & \cos 90 \cos \theta_1 & -\sin 90 \cos \theta_1 & 0 \\ 0 & \sin 90 & \cos 90 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$${}^1A_2 = \begin{pmatrix} \cos \theta_2 & -\cos 0 \sin \theta_2 & \sin 0 \sin \theta_2 & L_1 \cos \theta_2 \\ \sin \theta_2 & \cos 0 \cos \theta_2 & -\sin 0 \cos \theta_2 & L_1 \sin \theta_2 \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$${}^2A_3 = \begin{pmatrix} \cos \theta_3 & -\cos 0 \sin \theta_3 & \sin 0 \sin \theta_3 & L_2 \cos \theta_3 \\ \sin \theta_3 & \cos 0 \cos \theta_3 & -\sin 0 \cos \theta_3 & L_2 \sin \theta_3 \\ 0 & \sin 0 & \cos 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Relaciones geométricas

Al tratarse de un brazo articulado de dos eslabones los cálculos se reducen de manera considerable. Por este motivo ha sido el modelo elegido para el cálculo de la posición del extremo del brazo articulado.

Partiendo de los siguientes modelos en dos dimensiones, se pueden obtener las ecuaciones a resolver para conocer la situación del extremo del brazo respecto el eje X e Y de la base.

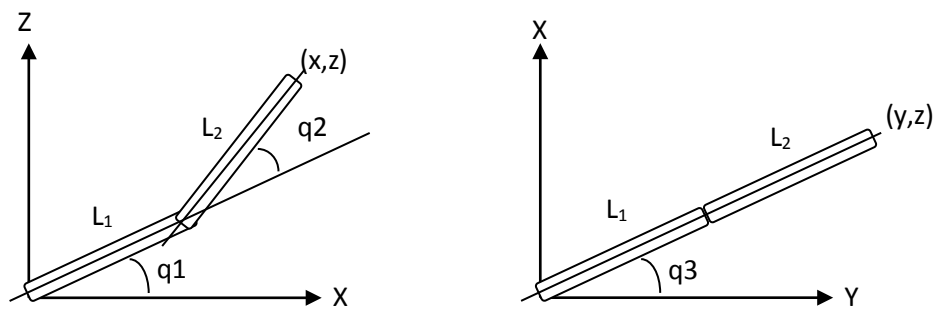


Ilustración 40. Relaciones geométricas- cinemática directa

$$x = L_1 \cdot \cos q_1 + L_2 \cdot \cos(q_1 + q_2) \quad [11]$$

$$z = L_1 \cdot \sin q_1 + L_2 \cdot \sin(q_1 + q_2) \quad [12]$$

$$y = (L_1 + L_2) \cdot \cos q_3 \quad [13]$$

6.2. Cálculo del modelo cinemático inverso

La resolución del modelo cinemático inverso, debe permitir encontrar los valores que debe adoptar cada una de las articulaciones de las que consta el brazo, para situar su extremo en un determinado punto. Con esta finalidad y siguiendo el modelo geométrico que se ha decidido utilizar para el cálculo del modelo cinemático directo, se obtienen las siguientes ecuaciones que permitirán posicionar la pinza en el sitio deseado.

Como el desplazamiento se realizará a través de los ejes, a partir de aquí se llamará al desplazamiento en los ejes X, Y, y Z como Δx , Δy , Δz siendo la variación entre la posición inicial y final de cada eje.

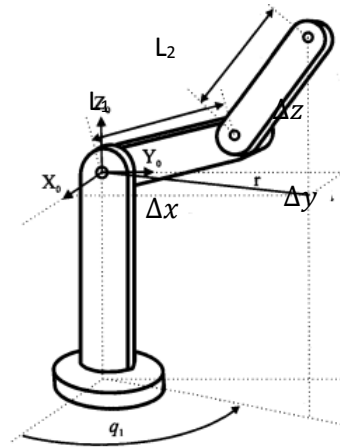


Ilustración 41. Modelo cinemático inverso

$$q_1 = \arctan\left(\frac{\Delta x}{\Delta z}\right) \quad [14]$$

Se puede obtener el siguiente sistema de ecuaciones:

$$r^2 = \Delta x^2 + \Delta y^2$$

$$r^2 + \Delta z^2 = l_1^2 + l_2^2 + 2l_1l_2 \cos q_3$$

Dónde:

$$q_3 = \arctan\left(\frac{\pm\sqrt{1-\cos^2 q_3}}{\cos q_3}\right) \quad [15]$$

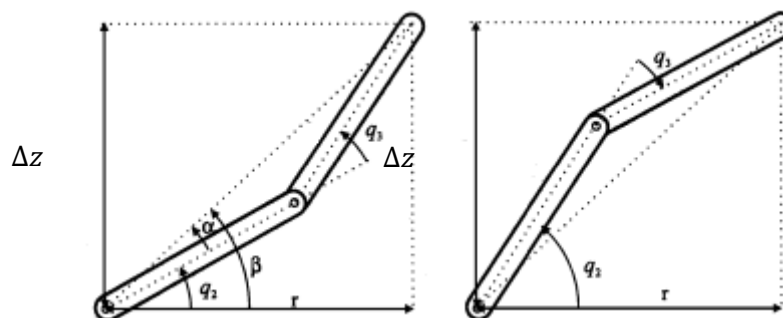


Ilustración 42. Relaciones geométricas-cinemática inversa

Se obtiene que:

$$q_2 = \beta - \alpha$$

$$\beta = \arctan\left(\frac{\Delta_z}{\pm\sqrt{\Delta_x^2 + \Delta_y^2}}\right); \alpha = \arctan\left(\frac{l_2 \sin q_3}{l_2 + l_3 \cos q_3}\right)$$

Resultando:

$$q_2 = \beta - \alpha = \arctan\left(\frac{\Delta_z}{\pm\sqrt{\Delta_x^2 + \Delta_y^2}}\right) - \arctan\left(\frac{l_2 \sin q_3}{l_2 + l_3 \cos q_3}\right) \quad [16]$$

7. Diseño electrónico

7.1. Selección de componentes

Una vez observadas las diferentes tipologías de brazo robótico con más apoyo y reconocimiento, es necesario comprender más a fondo su principio de funcionamiento y los componentes que interactúan para desarrollar un brazo robótico con criterio.

Debido al claro sentido práctico del proyecto y la inversión moderada que se quiere destinar, sólo se analizarán aquellos componentes más extendidos en las aplicaciones de modelismo de bajo coste.

Así pues, el sistema de comunicación más básico y eficiente para estas soluciones se trata de la interacción entre placa de control, servomotores y sus potenciómetros internos juntamente con la conexión Bluetooth para dar las ordenes necesarias al robot y la aplicación Android.



7.1.1. Placas de control

Una placa de control es una tarjeta electrónica con un microcontrolador especializado en el control de equipos que consta, principalmente, de una CPU (unidad de control de procesamiento), una memoria, unidades de entrada y salida y una memoria ROM que permite guardar el programa.

Los microcontroladores (abreviados como μC , uC o MCU), gracias a sus características reducidas en cuanto a coste y dimensiones, hacen que sean ideales para el control digital de gran cantidad de dispositivos.

Para la selección de la placa de control que se usará para el control del brazo robot se plantearon inicialmente dos placas de control de las marcas: Arduino y Raspberry. Ambas placas tienen semejanzas entre ellas, por un lado, en cuanto a Arduino su placa más común es Arduino UNO. En cuanto a Raspberry es Raspberry Pi. Las características principales de cada una de las placas presentadas se resumen en la siguiente tabla:

Tabla 14. Comparativa Arduino Uno VS Raspberry Pi (B)

		
	Arduino UNO	Raspberry Pi (B)
Precio (€)	21	25
Tamaño	7.6 x 1.9 x 6.4 cm	8.6 x 5.4 x 1.7 cm
Memoria SRAM:	2 KB	512 MB
Velocidad de reloj	16 MHz	700 MHz
On board Network	Ninguna	10/100 wired Ethernet RJ45
Voltaje de entrada	7 a 12 V	5 V
Memoria Flash	32 KB	Tarjeta SD (2 a 16G)
Puertos USB	Uno	Dos
Sistema operativo	Ninguno	Distribuciones de Linux
Entorno de desarrollo integrado (IDE)	Arduino	Scratch, IDLE, cualquiera con soporte Linux

Los motivos por los que se ha pensado en estos dos fabricantes de placas de control, son los siguientes:

1. Son libres y extensibles.

Se puede ampliar y modificar el diseño hardware de las placas como el entorno de desarrollo software y el propio lenguaje de programación.

2. Tienen una gran comunidad.
Son placas muy utilizadas y en muchos ámbitos. Esto significa que hay acceso rápido a gran cantidad de información.
3. Su entorno de programación es multiplataforma.
Se puede ejecutar e instalar en cualquier sistema operativo.
4. Precio económico.
Cómo ya se ha mostrado en la tabla anterior tienen un precio muy competitivo.

La diferencia principal, entonces, entre las dos placas presentadas es la capacidad de cómputo. Como se observa en la tabla 14, la velocidad de reloj de la placa de Arduino Uno es de 16 MHz, sin embargo, la velocidad de reloj de Raspberry Pi es de 700 MHz y lo que quiere decir, mucho más rápida.

Finalmente, considerando que para el control del brazo articulado es más importante el número de entradas y salidas digitales que la velocidad de reloj de su microcontrolador, se ha decidido utilizar una placa de la familia de Arduino.

A partir de aquí, se estudiará qué es y qué ofrece la plataforma Arduino.

Arduino es una plataforma de prototipos de electrónica de código abierto (open-source) basada en hardware i software flexible y fácil de usar.

En cuanto a software Arduino tiene una interface tal y como la que se muestra a continuación dónde se puede escribir el código necesario para la aplicación y que se subirá directamente a la placa conectada a cualquiera de los puertos USB de los que dispone el ordenador.

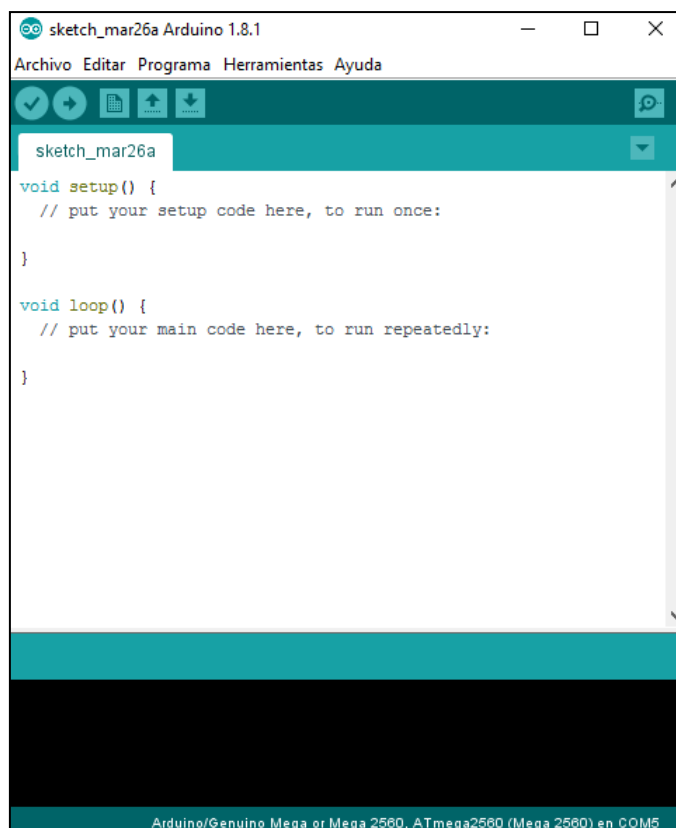


Ilustración 43. IDE Arduino

Por otro lado, en cuanto a hardware, Arduino dispone de gran variedad de placas que según sus características se adaptarán más a unos tipos de aplicación que a otras.

Entre ellas están Arduino UNO, Arduino MEGA 2560, Arduino LEONARDO, Arduino YUN, Arduino DUE etc.

Del conjunto de placas comentadas, la que se usará para el control del brazo robótico es la Arduino Mega 2560. Ha sido seleccionada ya que es la que ofrece más cantidad de entradas y salidas, analógicas y digitales. Concretamente, consta de 54 pines de entradas/salidas digitales, (14 de las cuales puedes ser utilizadas como salidas PWM), 16 entradas analógicas, 4 UARTs (puertos serie por hardware), cristal oscilador de 16 MHz, conexión USB, jack de alimentación, conector ICSP y botón de reset. Su microcontrolador es el ATmega2560 del fabricante Atmel.

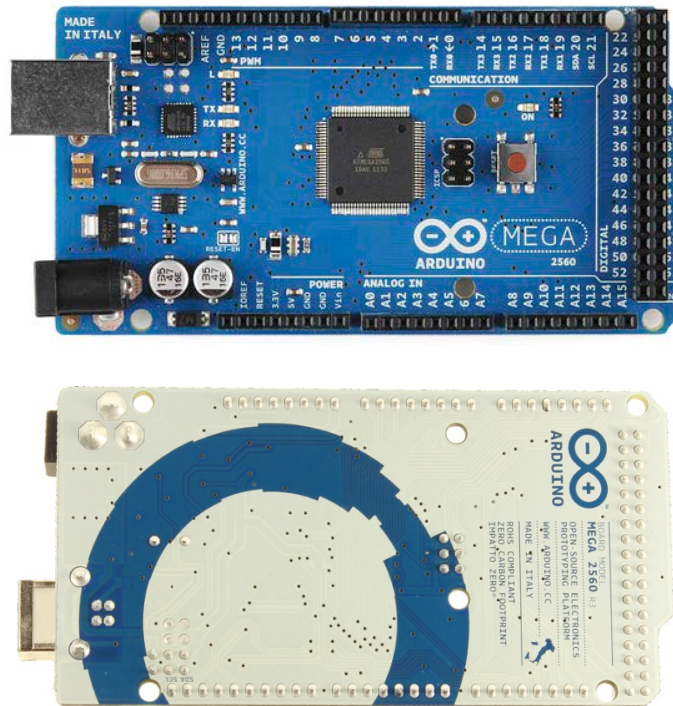


Ilustración 44. Arduino Mega 2560

Principales características:

Nombre: Mega 2560.

Procesador: ATmega2560.

Voltaje de entrada/funcionamiento: 3.8-5 V/3.3 V.

Velocidad CPU: 16 MHz.

Entradas/Salidas analógicas: 16/0.

Entradas/Salidas digitales: 54/15.

EEPROM [kB]: 4.

SRAM [kB]: 8.

Flash[kB]: 256.

USB: Regular.

UART: 4.

7.1.2. Servomotores

Actualmente los brazos robóticos más potentes cuentan con unos sistemas hidráulicos capaces de elevar cargas de más de una tonelada en un alcance máximo de 3 metros. Como es de esperar estos sistemas sólo se encuentran incorporados en cadenas de producción de grandes industrias. Sin embargo hay que ser consciente del compromiso existente entre el coste de los componentes y el régimen operativo final, el cual no es comparable a la industria. Esta consideración hace que se haya desarrollado una tecnología paralela al sector industrial: la tecnología aplicada al modelismo.

Un servomotor de modelismo es un actuador rotatorio de corriente continua que tiene la capacidad de ubicarse en cualquier posición dentro de su rango de operación, y mantenerse estable dentro de ésta.

Los servomotores son los componentes que protagonizan el movimiento de las articulaciones en los brazos robóticos de modelismo.

Anteriormente, el rango de valores máximos que podía alcanzar un servomotor era de unos aproximados 180º, conforme ha avanzado la técnica actualmente existen ya de hasta 360º.

En el alcance del presente trabajo, se estudiará el principio de funcionamiento de unos servomotores muy integrados en el sector de la robótica como son los servomotores de modelismo (Radio Control Servos), caracterizados por sus dimensiones reducidas, bajo consumo energético y fácil adquisición.



Ilustración 45. Ejemplo servomotor

Los servomotores de modelismo están conformados por 3 elementos principales:

- Un motor de corriente continua
- Un sistema de control que dispone de una pequeña placa de circuito impreso (PCB) y un potenciómetro (acoplado al eje de salida).
- Una caja reductora que aumenta el par transmitido por el motor al eje de salida.

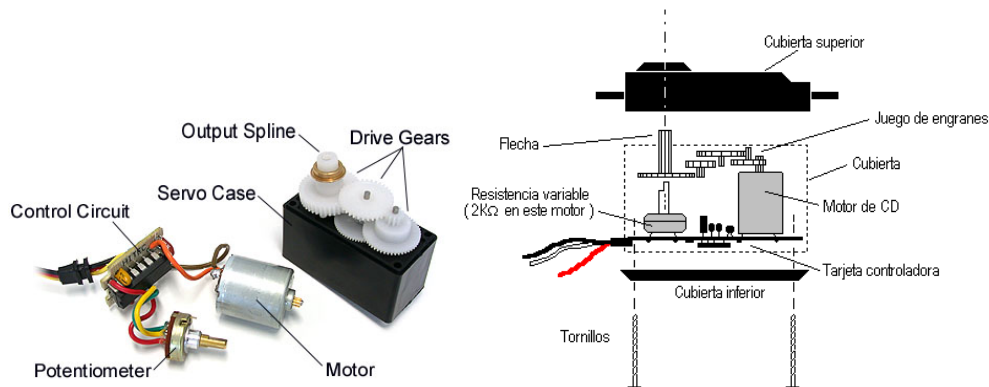


Ilustración 46. Elementos principales de un servomotor

Su modo de conexión se realiza a partir de 3 cables estandarizados donde el voltaje de alimentación es generalmente de entre 4,8 y 6 voltios. Este valor de voltaje siempre viene limitado por el circuito de control, mucho más restrictivo que el motor. Según la marca del fabricante, los colores de los cables suelen caracterizarse en:

- Cable rojo: Corresponde al terminal positivo de alimentación
- Cable negro / marrón: Corresponde al terminal negativo de alimentación también llamado tierra.
- Cable naranja / amarillo / blanco: Corresponde al terminal de entrada de la señal de control.

Los servomotores hacen uso de la modulación por ancho de pulsos eléctricos (PWM o Pulse-Width-Modulation) para controlar la dirección o posición de los motores de corriente continua. La mayoría de ellos trabajan en la frecuencia de 50 Hz, esto es pues un periodo de 20 milisegundos.

Como se presenta en la ilustración 47, según el ancho de pulso que se configure al cable de control, el servomotor se posicionará en el ángulo correspondiente.

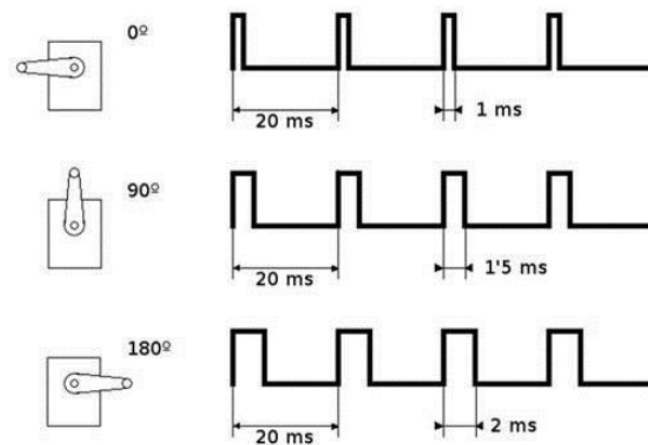


Ilustración 47. Señal PWM para el control del servomotor

Es importante ser consciente de que el proceso que se da lugar desde la introducción de la señal hasta el posicionamiento del servomotor no es directo, pues el componente más primitivo y esencial del sistema, el motor de corriente continua, no tiene capacidad de autorregularse. El motor CC girará en mayor o menor velocidad y en un sentido u otro según la magnitud y signo de la diferencia de potencial aplicada a sus bornes.

Para controlar el posicionamiento de este elemento, los servomotores tienen diseñados internamente un sistema de control en lazo cerrado similar al esquematizado en la ilustración 48.

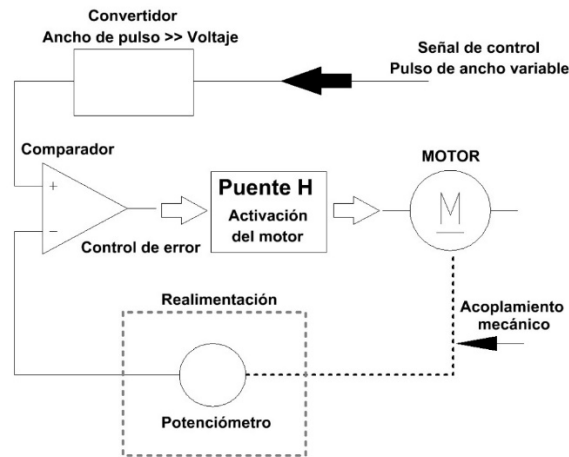


Ilustración 48. Sistema de control interno del servomotor

Gracias a este sistema, el servomotor en todo momento sabe en qué posición se encuentra, ya que tiene acoplado un potenciómetro en el eje de salida que suministra un voltaje proporcional a la posición angular.

Si en algún momento la señal de entrada PWM (convertida a voltaje por un filtro pasa-baja) y por tanto el valor angular de entrada difiere del valor real, se calculará la diferencia necesaria para alcanzar el ángulo en forma de voltaje mediante un comparador (error angular). Este valor será enviado al motor, siguiendo un proceso iterativo hasta llegar a un error angular nulo, y por tanto, el correcto posicionamiento del servomotor y apagado.

Con este sistema también se puede detectar si se está forzando mecánicamente el servomotor fuera de su posición.

Si se entra más en el detalle de este sistema de control, la configuración más utilizada en servomotores es la estructura en puente H.

Una vez descrito el principio de funcionamiento de los servomotores, a continuación se presentan los que han sido elegidos, mediante cálculos mecánicos que se pueden encontrar en la anterior sección, para soportar i mover la estructura del prototipo.

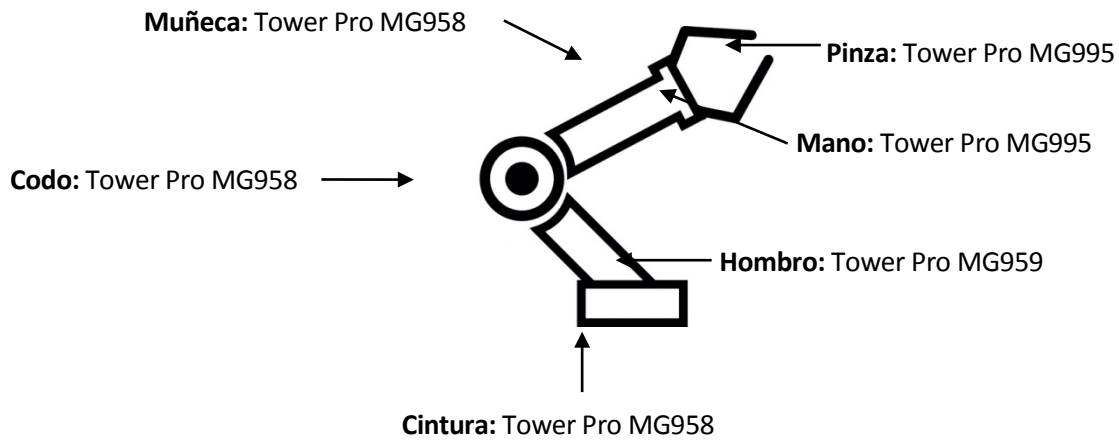


Ilustración 49. Selección servomotores



Ilustración 50 . TowerPro MG995



Ilustración 51. TowerPro MG958

Tabla 15. Tabla características TowerPro MG995

TOWER PRO MG995	
Voltaje	4.8 V– 6.0V
Torque (4.8 V)	10.0 kg·cm
Torque (6.0 V)	11.00 kg·cm
Carcasa	Aluminio
Piñonería:	Metálica
Rodamientos:	2
Rango de temperatura	0-55 °C
Temperatura de funcionamiento	-20 - +60 °C
Velocidad (4.8 V)	0.20s/60º
Velocidad (6.0 V)	0.16s/60º
Longitud del cable	32 cm

Tabla 16. Tabla características TowerPro MG958

TOWER PRO MG958	
Voltaje	4.8 V– 6.6V
Torque (4.8 V)	18.0 kg·cm
Torque (6.6 V)	20.0 kg·cm
Carcasa	Aluminio
Piñonería:	Metálica
Rodamientos:	2
Rango de temperatura	0-55 °C
Temperatura de funcionamiento	-20 - +60 °C
Velocidad (4.8 V)	0.18s/60º
Velocidad (6.6 V)	0.15s/60º
Longitud del cable	32 cm



Ilustración 52. Tower Pro MG959

Tabla 17. Tabla características TowerPro MG959

TOWER PRO MG959	
Voltaje	6.0 V– 7.4 V
Torque (6.0 V)	28.0 kg·cm
Torque (7.2 V)	32.0 kg·cm
Carcasa	Aluminio
Piñonería:	Metálica
Rodamientos:	2
Rango de temperatura	0-55 °C
Temperatura de funcionamiento	-20 - +60 °C
Velocidad (6.0 V)	0.19s/60º
Velocidad (7.2 V)	0.17s/60º
Longitud del cable	32 cm

7.1.3. Bluetooth

Se define Bluetooth, como un protocolo de comunicación para redes inalámbricas que permite la transmisión de voz y datos entre distintos dispositivos mediante una radiofrecuencia segura (2,4 GHz). Es una forma de intercambiar información entre dispositivos como pueden ser ordenadores, teléfonos móviles, impresoras etc. de bajo coste y a través de ondas de radio de baja frecuencia.



Ilustración 53. Logotipo Bluetooth

Su origen se sitúa en la década de los 90' cuando la marca Ericsson estaba desarrollando una tecnología que permitiera comunicaciones de corto alcance y con un muy bajo consumo. Posteriormente con el gran interés que pusieron las grandes compañías del momento como eran Apple, Ericsson, Intel, Lenovo, Microsoft, Motorola, Nokia, Nordic Semiconductor y Toshiba, crearon un grupo de trabajo llamado SIG(Special Interest Group) con el fin de acabar creando el protocolo que actualmente estamos usando para realizar infinidad de comunicaciones.

Su principio de operación se basa en el modo maestro/esclavo. Se utiliza la palabra "piconet" (ilustración 54) para hacer referencia a la red formada por un dispositivo y todos los dispositivos que se encuentran dentro de su rango. Un dispositivo que tiene el rol de maestro se puede conectar simultáneamente con hasta siete dispositivos esclavos. Cada uno de los dispositivos de una piconet posee una dirección lógica de 3 bits.

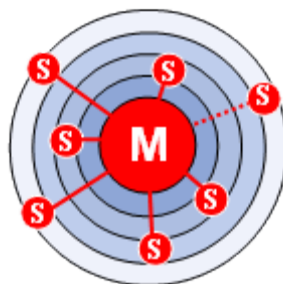


Ilustración 54. Piconet

El dispositivo maestro sólo puede conectarse con un solo esclavo al mismo tiempo, por esto, cambia de manera rápida entre esclavos para que parezca que se está conectando simultáneamente con todos los dispositivos esclavos.

El establecimiento de las conexiones sigue un proceso como el que se presenta a continuación:

Modo pasivo: todos los dispositivos se encuentran en modo pasivo dentro de la red, es decir, están esperando a ser llamados por un dispositivo maestro.

Solicitud: cuando el maestro quiere iniciar una comunicación envía una solicitud a todos los dispositivos, una vez los dispositivos reciben la solicitud responden con su dirección.

Paginación: una vez el maestro ha recibido todas las direcciones de los dispositivos que se encuentran en su red, elige una de las direcciones con la que realiza una sincronización con su reloj y frecuencia.

Descubrimiento del servicio: cuando la anterior fase se da por finalizada, se establece un enlace con el punto de acceso que permite al dispositivo maestro ingresar a una fase de descubrimiento del servicio del punto de acceso mediante un protocolo denominado SDP (Protocolo de Descubrimiento de Servicios). Cuando esta fase acaba, los dos dispositivos inician un canal de comunicación.

Emparejamiento: en ocasiones, el punto de acceso, puede incluir sistemas de seguridad de emparejamiento. Esto es necesario ya que es una manera de asegurar que sólo tienen acceso a la comunicación usuarios autorizados. El emparejamiento se realiza con una clave "PIN". En este caso, el dispositivo maestro recibe una solicitud para ingresar el código PIN del punto de acceso. En caso de que el PIN sea correcto se habrá establecido de manera segura la conexión.

En este proyecto, para realizar este tipo de comunicación entre un dispositivo móvil y el brazo robot articulado se usará un módulo Bluetooth de bajo coste llamado HC-06.

El módulo HC06, es un módulo Bluetooth muy popular para aplicaciones con microcontroladores tal y como Arduino o PIC. Se caracterizan por ser relativamente económicos y habitualmente se comercializan en un formato que permite cablearlos fácilmente con la placa de control.



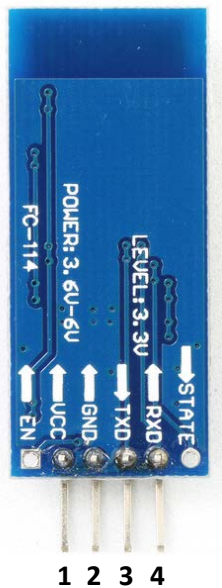
Ilustración 55. Módulo Bluetooth HC-06

Sus características principales se adjuntan a continuación:

- *Compatible con el protocolo Bluetooth V2.0.*
- *Voltaje de alimentación de 3.3Vdc a 6 Vdc.*
- *Voltaje de operación de 3.3 Vdc.*
- *Velocidad de Baudios: 1200, 2400, 4800, 9600, 19200, 38400, 57600 y 115200.*

- **Tamaño:** 4.4 x 1.6 x 0.7 cm.
- **Corriente de operación:** < 40 mA.
- **Corriente en modo sleep (sin transmisión de datos):** <1 mA.

Está formado por cuatro pines de conexión. En la siguiente ilustración se puede observar su patillaje de conexión:



- 1 – **VCC:** Tensión de alimentación del módulo.
- 2 – **GND:** Conexión a tierra.
- 3 – **TXD:** Transmisión de datos.
- 4 – **RXD:** Recepción de datos.

Ilustración 56. Patillaje conexión módulo HC-06

Este módulo puede actuar tanto como esclavo como maestro y se puede encontrar en el mercado por un precio de aproximadamente 7 €.

7.1.4. Android

Android es un sistema operativo (OS) inicialmente creado para dispositivos móviles, que se diferencia de otros sistemas tal y como podrían ser iOS, Symbian y Blackberry OS por el hecho de que es un sistema operativo libre, gratuito y multiplataforma.



Ilustración 57. Logotipo Android

A su vez, Android, proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono de una forma sencilla y en un lenguaje de programación conocido como es el Java. Esto hace que existan infinidad de aplicaciones para este sistema operativo al alcance de cualquier usuario.

Éste OS no se hizo popular hasta 2005 que la compañía Google lo compró. Aunque inicialmente su progreso fue relativamente lento, en 2010 se coronó como el sistema operativo más vendido en el mundo.

El motivo principal por el cual se ha decidido crear una aplicación para un dispositivo Android para el control y manejo del brazo articulado es la libertad que ofrece dicho sistema operativo. Ni programar en este sistema ni incluirlo en un teléfono móvil supone un coste, ya que es completamente gratuito.

7.2. Componentes electrónicos del brazo articulado

Los componentes electrónicos necesarios para el control del brazo robot son:

- **3 servomotores Tower Pro MG958.**
Serán los servomotores que se encargaran del movimiento de la cintura, codo y muñeca. Se trata de unos servomotores de par alto y por tanto se sitúan en las articulaciones que se necesita más par. Los tres se sitúan en la base.
- **1 servomotor Tower Pro MG959.**
Se encargará del movimiento del hombro y estará situado en la base. Es el que tiene el par más alto de todos.
- **2 servomotores Tower Pro MG995.**
Se encargarán del movimiento de la pinza y mano. Uno de ellos servirá para abrir y cerrar la pinza mientras que el otro se encargará de girarla.
- **1 placa Arduino Mega 2560.**
Es la placa de control seleccionada para el proyecto y dónde se cargará el programa.
- **6 diodos IN4004.**
Son unos rectificadores de silicona que se situarán entre la salida de la señal PWM generada en el pin digital de Arduino y la entrada del pin de control PWM del servomotor.
- **1 condensador de 1 μ F.**
Se utilizará para estabilizar la señal de la fuente de alimentación. Se posicionará entre el positivo y el negativo de la fuente.

- **1 Módulo Bluetooth HC-06.**

Con él se realizará la comunicación y el intercambio de datos entre la aplicación Android y la placa de control Arduino.

- **1 fuente de alimentación.**

Se utilizará una fuente de alimentación extraída de la torre de un ordenador antiguo que servirá para proporcionar el voltaje y la tensión necesaria para mover el brazo.

Las características de la fuente de alimentación son:

- Marca: Mercury.
- 300W.
- Voltaje de entrada: 115/220 Vac.
- Frecuencia: 50-60 Hz.
- Voltaje salida: +3.3V (28 A), +5V (30A), +12V (15A), -5V (0.3A), -12V (0.8A).

7.3. Esquema de conexión

El esquema de conexión del conjunto de componentes que forman la parte electrónica del brazo articulado se ha diseñado mediante el programa Fritzing. Se trata de un software de descarga gratuita y muy fácil de usar. A la vez, es el más recomendado cuando se trata de hacer esquemas de conexión que implican placas de control Arduino debido al amplio abanico de librerías y componentes que ofrece.

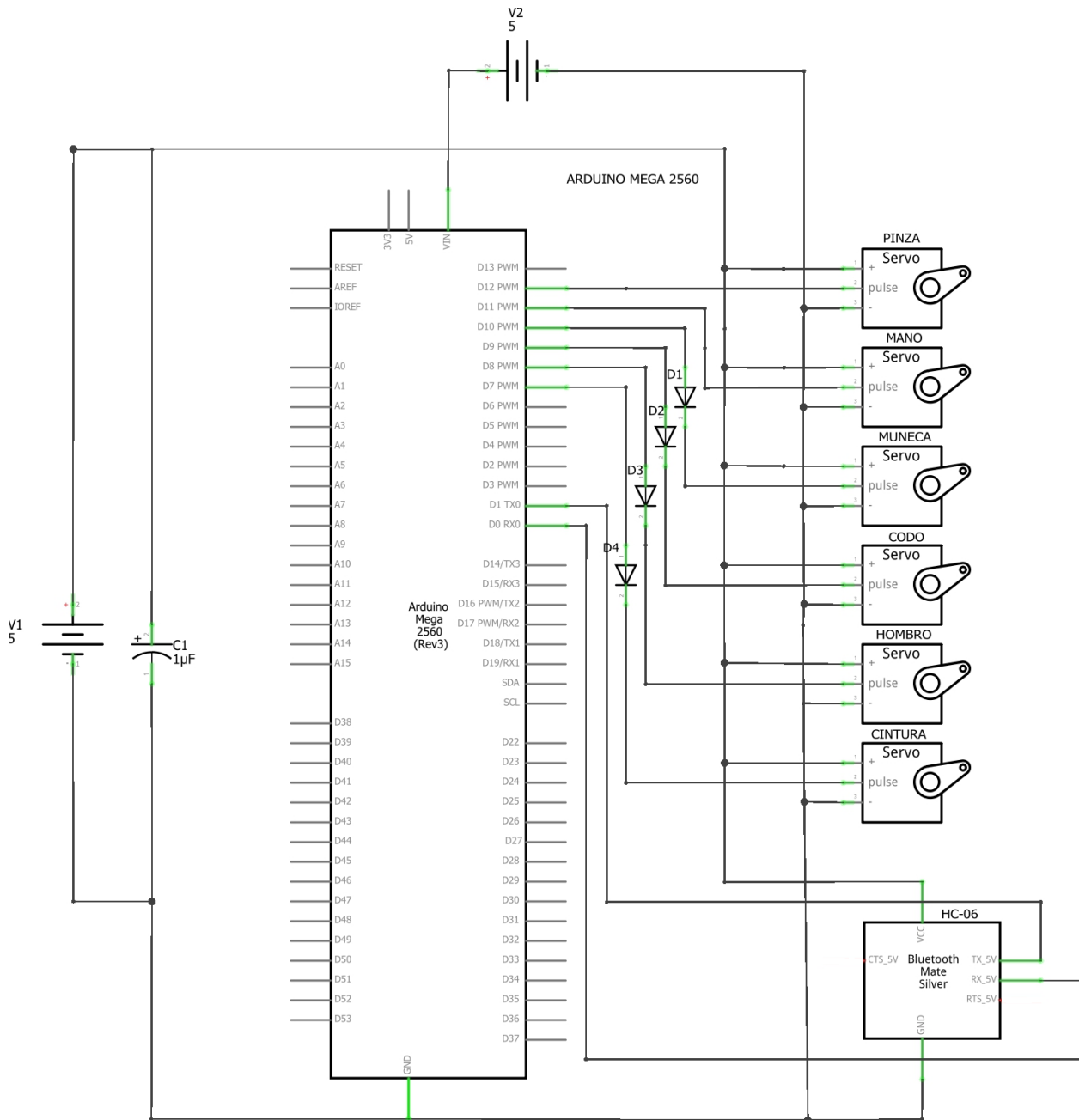


Ilustración 58. Esquema de conexión electrónico

Un timer o interrupción programada tiene la misma función que la que podría tener una interrupción hardware. En este caso los timers nos servirán para dar un aviso cada cierto período programado sin tener la necesidad de controlar el tiempo.

En el caso de Arduino Mega 2560, tiene un cristal que “bate” a 16 MHz, o lo que sería lo mismo, 16.000.000 veces por segundo. Esto entonces, nos indica que podríamos tener una interrupción cada 0,000000006 segundos aunque en realidad esto no sucede así ya que cada instrucción de Arduino necesita varios pulsos de reloj para ejecutarse.

Cada timer tiene un registro interno que indica cada cuantos flancos de reloj debe dispararse. Cómo es probable que se necesite flexibilidad para los tiempos los microcontroladores que forman las placas de Arduino incluyen divisores de la frecuencia básica del cristal que permitirán esta flexibilidad.

Así pues, para no tener que usar la función delay que podría paralizar el programa por momentos, la señal PWM será generada mediante interrupciones que harán que cambie el estado del pin de alto a bajo o viceversa y de este modo creará la señal modular.

Arduino Mega 2560 dispone de seis timers y quince salidas PWM:

Timer 0: controla a los pines 4 y 13.

Timer 1: controla a los pines 11 y 12.

Timer 2: controla a los pines 9 y 10.

Timer 3: controla a los pines 2, 3 y 5.

Timer 4: controla a los pines 6, 7 y 8.

Timer 5: controla a los pines 44, 45 y 46.

Analizando el esquemático presentado anteriormente se puede observar que se están usando los timers 1, 2 y 4 para seguir el orden de los pines digitales en la placa de control. Sin embargo, el motivo por el cual el pin 13 (timer 0) no ha sido usado, es por qué la transmisión de datos (Tx y Rx) en el que se ha conectado el módulo Bluetooth HC-06 utiliza este timer para realizar la sincronización con el envío y la recepción de información.

También es importante apuntar que los timers 1, 2, 3, 4, 5 trabajan a una frecuencia de 500 Hz, sin embargo el timer 0 trabaja a una frecuencia de 1000 Hz.



8. Protocolo de comunicaciones

La comunicación entre el módulo Bluetooth (Arduino) y la aplicación móvil se consigue mediante el envío de paquetes de datos. Al pulsar los diferentes botones en la aplicación se envía un paquete de datos mediante la comunicación inalámbrica, que Arduino recibirá y decodificará para entender la instrucción que se le ha pedido.

Para evitar cualquier problema, se ha decidido fijar una cabecera i un final (ambos de dos caracteres) que será el mismo en todas las instrucciones. Esto permitirá saber cuándo empieza una instrucción y cuándo termina.

Todas las instrucciones que reciba el programa de Arduino procedentes de la aplicación móvil serán con la siguiente cabecera de dos caracteres: "AA". A su vez, todas las instrucciones finalizarán con un final, también formado por dos caracteres: "BB".

AA	Código de la instrucción	BB
----	--------------------------	----

A partir de aquí, los códigos para cada una de las diferentes instrucciones son los siguientes:

Instrucciones TEACH:

AA	Instrucción teach	BB
----	-------------------	----

Este tipo de instrucciones permitirán mover el extremo del brazo articulado de dos modos diferentes: por articulaciones (JOINT) y por coordenadas (X, Y, Z).

Movimiento por articulaciones (JOINT):

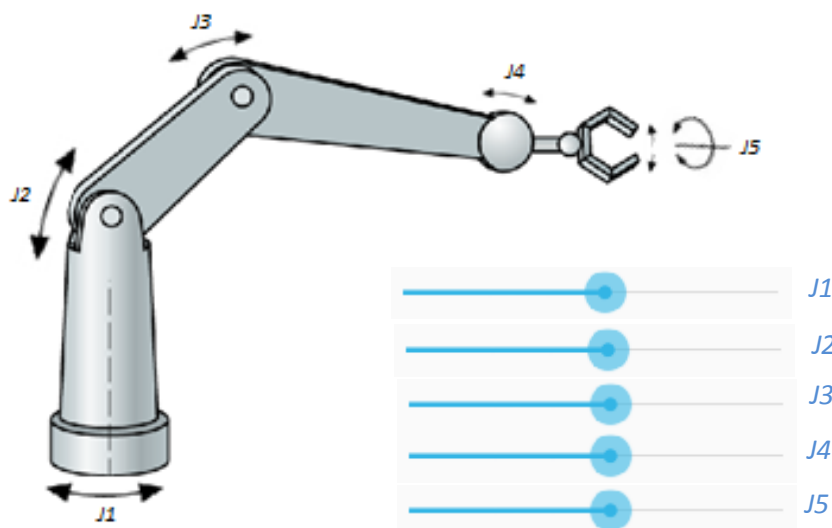


Ilustración 59. Esquema movimiento JOINT

Movimiento por articulaciones (X, Y, Z):

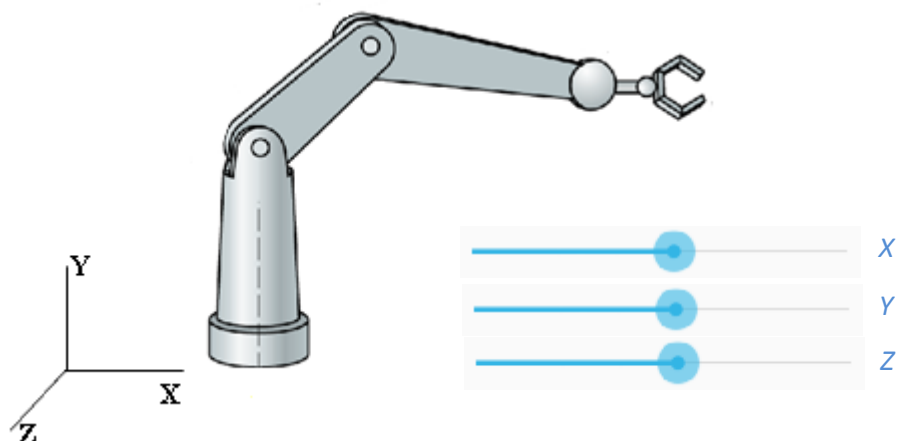


Ilustración 60. Esquema movimiento X, Y, Z

La instrucción teach estará formada por cuatro caracteres:

Primer carácter	Segundo carácter	Tercer carácter	Cuarto carácter
-----------------	------------------	-----------------	-----------------

Primer carácter: el primer carácter de las instrucciones teach sólo podrá tener 3 valores diferentes: 1, 2 y 3.

- Caso primer carácter=1:

Si el primer carácter de la instrucción teach es un 1 significará que se está enseñando al robot a través de un movimiento por coordenadas. Es decir que el usuario está indicando que quiere un desplazamiento a través de uno de los tres ejes.

- Caso primer carácter=2:

Si el primer carácter de la instrucción teach es un 2 significará que se está enseñando al robot a través de un movimiento por articulaciones. Es decir que el usuario está indicando que quiere el desplazamiento de una de las articulaciones del brazo.

- Caso Primer carácter=3:

Si el primer carácter de la instrucción teach es un 3 querrá decir que el usuario quiere guardar una posición.

A partir de aquí seguiremos con el segundo carácter que dependerá del primero y así sucesivamente.

Segundo carácter:

- Si primer carácter=1:
 - Si segundo carácter=1: indica desplazamiento a través del eje de coordenadas X.
 - Si segundo carácter=2: indica desplazamiento a través del eje de coordenadas Y.
 - Si segundo carácter=3: indica desplazamiento a través del eje de coordenadas Z.
- Si primer carácter =2:
 - Si segundo carácter=1: movimiento de la articulación de la cintura.
 - Si segundo carácter=2: movimiento de la articulación del hombro.
 - Si segundo carácter=3: movimiento de la articulación del codo.
 - Si segundo carácter=4: movimiento de la articulación de la muñeca.
 - Si segundo carácter =5: movimiento de la articulación de la mano.
- Si primer carácter=3:
 - Si segundo carácter=1: guardar posición 1.
 - Si segundo carácter=2: guardar posición 2.
 - Si segundo carácter=3: guardar posición 3.
 - Si segundo carácter=4: guardar posición 4.

Tercer carácter:

- Si primer carácter=1:
 - Si tercer carácter=0: desplazamiento en el eje hacia abajo.
 - Si tercer carácter=1: desplazamiento en el eje hacia arriba.
- Si primer carácter=2:
 - Si tercer carácter=0: desplazamiento de la articulación hacia la derecha.
 - Si tercer carácter=1: desplazamiento de la articulación hacia la izquierda.
- Si primer carácter=3: tercer carácter siempre es igual a 2 para tener una longitud de paquete fija.

Cuarto carácter:

- Si primer carácter=1 ó 2:
 - Si cuarto carácter=1: longitud de desplazamiento baja.
 - Si cuarto carácter=2: longitud de desplazamiento media.
 - Si cuarto carácter=3: longitud de desplazamiento alta.
- Si primer carácter=3: tercer carácter siempre es igual a 4 para tener una longitud de paquete fija.

Tabla resumen:

Tabla 18. Paquetes de datos pantalla teaching

Cabecera	Primer carácter	Segundo carácter	Tercer carácter	Cuarto carácter	Final
AA	1 (X,Y,Z)	1	0	1	BB
		2	1	2	
		3	-	3	
	2 (JOINT)	1	0	1	
		2	1	2	
		3	-	3	
		4	-	-	
	3 (GUARDAR POS)	1	2	4	
		2	-	-	
		3	-	-	
		4	-	-	

Instrucciones EXECUTOR:

AA	<i>Instrucción executor</i>	BB
----	-----------------------------	----

La instrucción teach estará formada por cinco caracteres:

4	<i>Primer carácter</i>	<i>Segundo carácter</i>	<i>Tercer carácter</i>	<i>Cuarto carácter</i>
---	------------------------	-------------------------	------------------------	------------------------

En este caso, se tiene que:

Primer carácter: primera posición a la que queremos que se desplace el brazo articulado al iniciar el recorrido. (Puede ser 0,1,2,3,4)

Segundo carácter: segunda posición del recorrido. (Puede ser 0,1,2,3,4)

Tercer carácter: tercera posición del recorrido. (Puede ser 0,1,2,3,4)

Cuarto carácter: cuarta posición del recorrido. (Puede ser 0,1,2,3,4)

Tabla resumen:

Tabla 19. Paquetes de datos pantalla executor

AA	4	0	0	0	0	BB
		1	1	1	1	
		2	2	2	2	
		3	3	3	3	
		4	4	4	4	

Instrucciones PINZA:

AA	<i>Instrucción pinza</i>	BB
----	--------------------------	----

La instrucción teach estará formada por cinco caracteres:

5	<i>Primer carácter</i>	<i>Segundo carácter</i>	<i>Tercer carácter</i>	<i>Cuarto carácter</i>
---	------------------------	-------------------------	------------------------	------------------------

Primer carácter: estado de la pinza en el punto 1. (0 cerrado, 1 abierto).

Segundo carácter: estado de la pinza en el punto 2.

Tercer carácter: estado de la pinza en el punto 3.

Cuarto carácter: estado de la pinza en punto 4.

Tabla resumen:

Tabla 20. Paquetes de datos pinza

AA	5	0	0	0	0	BB
		1	1	1	1	

9. Programación del controlador del brazo robot

Para el diseño del software del microcontrolador que gobernará el funcionamiento del brazo robótico se han seguido los pasos básicos a la hora de escribir un programa. En primer lugar se realizará un diagrama de flujo. Un diagrama de flujo no es más que una representación gráfica que incluye todos los pasos de manera esquemática por los que puede pasar el programa.

Una vez preparado el diagrama de flujo se pasará a escribir el programa en el software de Arduino. Para ello ha sido necesario estudiar las estructuras de programación que ofrece a la vez que elegir las librerías ya predeterminadas que se van a usar en el programa.

9.1. Diagrama de flujo del programa

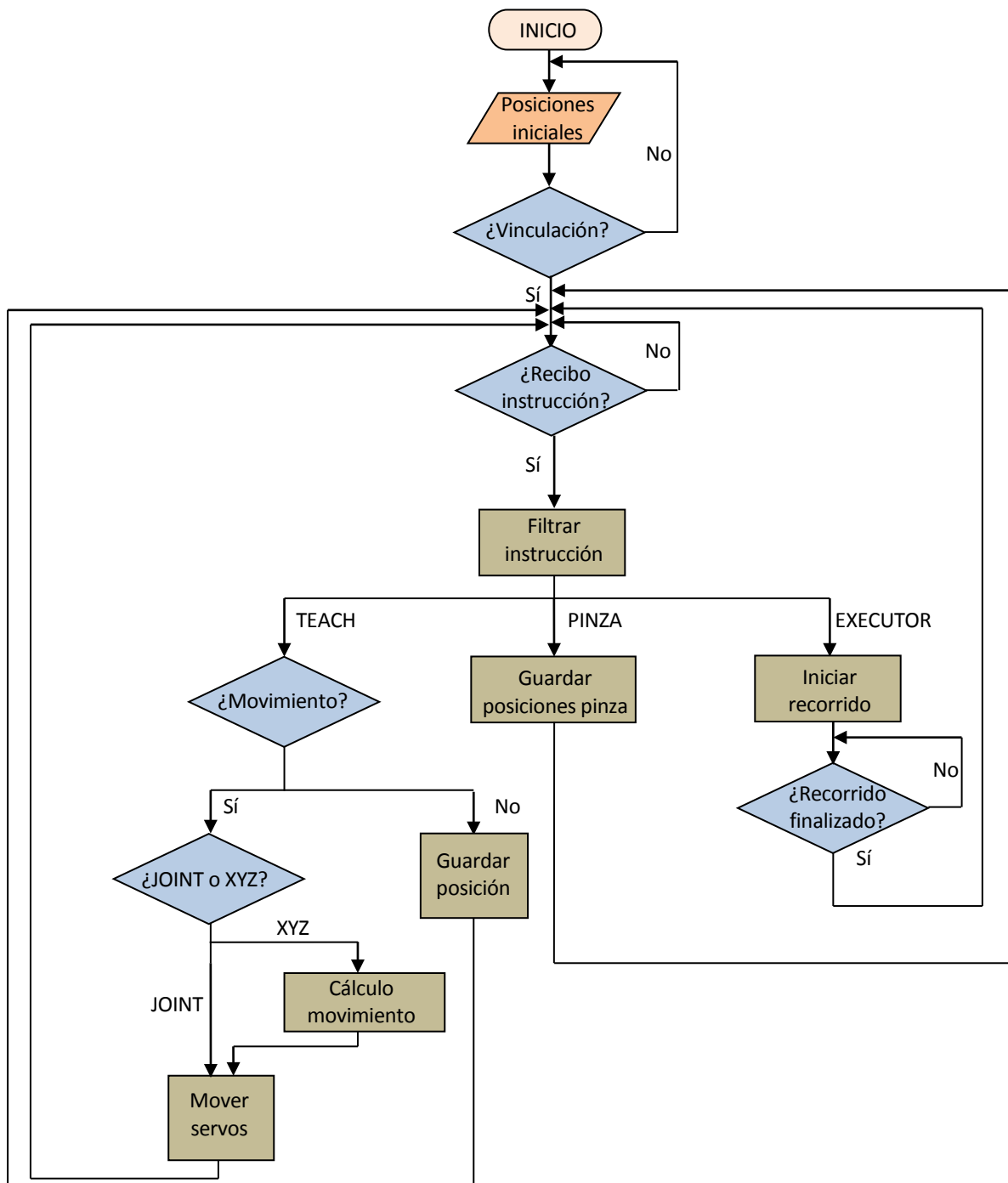


Ilustración 61. Diagrama de funcionamiento del programa de Arduino

Al alimentar el brazo robot, éste realizará la secuencia descrita en el diagrama de flujo.

En primer lugar se situará cada articulación en una posición preestablecida como inicial. La pinza, en su caso quedará abierta. Una vez alcanzadas las posiciones iniciales el programa quedará a la espera de recibir la orden de que la vinculación y conexión con el robot se ha iniciado.

Una vez se haya establecido la comunicación entre ambos componentes, el programa quedará a la espera de recibir instrucciones de la aplicación móvil a través del puerto serie dónde estará conectado el módulo Bluetooth.

En el momento en que se reciba una instrucción realizará un filtro para saber qué clase de instrucción está recibiendo. Podrá recibir tres tipos de instrucciones: instrucción teach, instrucción de posicionamientos de pinzas e instrucción executor.

Cuando reciba una instrucción teach el programa deberá diferenciar entre una instrucción teach de posicionamiento, es decir, de movimiento del robot (ya sea de modo JOINT o de modo X, Y, Z) o instrucción teach para guardar posición cuando ya se ha llevado el extremo a la posición deseada.

En caso de recibir una instrucción teach de posicionamiento por JOINT el movimiento será directo. Sin embargo, cuando se reciba una posición de carácter teach pero de movimiento X, Y, Z se utilizará una función que calculará el movimiento que deberá realizar cada motor para que el movimiento final sea el desplazamiento sobre uno de los tres ejes de coordenadas.

En el momento en que reciba una instrucción de pinza, lo único que deberá hacer el programa es almacenar la información para saber en qué modo deberá posicionar la pinza entre diferentes puntos (abierta o cerrada).

Por último, en caso de recibir una instrucción de recorrido, lo único que deberá hacer es posicionar cada uno de los motores en los puntos que forman el recorrido recibido. Ésta es la tarea que conlleva más tiempo y se podría decir más líneas de programación ya que se trata de una secuencia. Al iniciar el recorrido, los motores se moverán al mismo tiempo todos para conseguir la posición requerida, una vez hayan conseguido dicha posición, entonces se efectuará el cambio (o no) del estado de la pinza según haya indicado el usuario con la aplicación móvil y así sucesivamente.

9.2. Programación con Arduino

El entorno de programación (IDE) de Arduino sigue una estructura muy parecida al C y su orden está bien definido. Inicialmente se declaran las librerías que se van a usar en el programa. Seguidamente se inicia la declaración de variables y después proceden las estructuras del setup y loop. Las órdenes que aparecen en el setup sólo se llevan a cabo al iniciar el programa. En cambio, todo lo que se introduce en el loop, se reproducirá en la placa de modo infinito.

A continuación, siguiendo el orden de la estructura de cualquier programa en Arduino, se comentaran cada una de las partes y lo que suelen contener normalmente.

<Librerías>

Arduino dispone de una gran cantidad de librerías integradas de modo predeterminado. Su invocación y llamamiento se realiza con el siguiente comando:

```
#include <Nombredealibreria.h>
```

En el programa diseñado para el control del brazo articulado se ha utilizado la librería PWM.

Ésta librería permite la creación de señales PWM en los pines de salida digitales. Lo que hace es crear una onda cuadrada, es decir un señal interruptivo entre el estado de salida alto (5 Volts) y el bajo (0 Volts). Ésta señal permanece en estado alto durante un período de tiempo que se llama ancho de pulso. Este valor es el que deberemos cambiar para conseguir poder mover los servomotores. Es decir, se realizará su movimiento por modulación del ancho de pulso. El valor del ancho de pulso se modificará con la función que ofrece la librería: `analogWrite()`. Así mismo, este librería también permite fijar la frecuencia a la que se quiere emitir la señal PWM.

La siguiente ilustración muestra la señal que se puede generar con esta librería con diferentes anchos de pulsos.

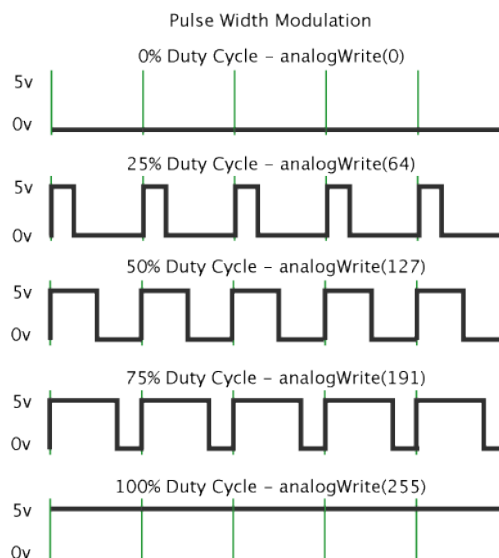


Ilustración 62. Generación de señal PWM en las salidas digitales de Arduino

Aunque normalmente la librería utilizada para el control de servomotores en Arduino es la librería Servo.h, en este proyecto se ha decidido no usarlo debido a que ésta librería viene acompañada siempre con delays. Un delay es un tiempo de duración determinada en el que se paraliza el programa y no realiza ninguna función. Para evitar esta problemática, en el programa se ha decidido crear la señal PWM sin usar esta librería.

Después de la librería, siguiendo la estructura del programa, viene la declaración de variables. El conjunto de variables básicas disponibles en Arduino son las que se resumen en la siguiente tabla.

Tabla 21. Variables Arduino

int	Valores de 16 bits (de -32768 a 32767)
unsigned int	Valores de 16 bits positivos (de 0 a 65535)
boolean	Valores de tipo cierto/falso
char	Valores alfanuméricos en forma de byte según ASCII
byte	Valores de 8 bits (de 0 a 255)
long	Valores de 32 bits (de -2147483648 a 2147483647)
unsigned long	Valores de 32 bits positivos (de 0 a 42949667295)
float	Valores decimales de 32 bits
array	Se trata de configuraciones de listas por medio de int o byte especificando al final el número de variables que almacenan entre claudators

Una vez definidas las variables, a continuación viene la función **void setup()** ésta función es llamada una sola vez por la placa des de su inicialización. En ella se definen las posiciones iniciales y se configuran pins, puertos série, inicialización de timers...

En el programa de control del brazo articulado se incluyen las siguientes funciones como las más importantes:

InitTimers(); Nos sirve para inicializar todos los timers que tiene la placa de control. En este caso, Arduino Mega 2560 dispone de cuatro timers.

analogWrite(pin, ancho de pulso); Cómo ya se ha comentado anteriormente esta función permite posicionar los servomotores en la posición inicial generando una señal de control en su pin PWM.

SetPinFrequency(pin, frecuencia); Ésta función permite fijar la frecuencia a la que se quiere generar la señal PWM. En este caso será la frecuencia del servomotor que se conecte al pin.

Una vez finalizado el **void setup()** la siguiente parte que forma todos los programas de Arduino es el **void loop()**. En el void loop se ponen las partes del programa que se quiere que se repita de manera infinita mientras el programa esté en ejecución. Todas las instrucciones que se incorporen en su interior serán entonces repetidas infinitas veces.

Dentro de ésta función se puede hacer el llamamiento a otras de creación libre según el usuario. Para la creación de nuevas funciones bastara con incluir la siguiente estructura después del void loop y hacer su llamamiento cuanto sea preciso.

Void loop()

```
{
    Nombredelafunción();
}
```

Void nombredelafunción ()

```
{
}
```

Por último comentar que Arduino también es capaz de fijar condicionales con las estructuras que se muestran a continuación:

Tabla 22. Funciones básicas Arduino

<pre>while (condición){ órdenes }</pre>	Siempre la condición se cumpla, se reproducirá el diálogo de órdenes interiores.
<pre>for (variable condición operación){ órdenes }</pre>	Siempre que la condición se cumpla se reproducirán las órdenes interiores del diálogo. En este caso se especifica la variable y la operación que percibirá cada vez que se reproduzca completamente el diálogo de órdenes interiores.
<pre>If (condición){ órdenes }</pre>	Si la condición se cumple, se reproducirá el diálogo de órdenes interiores una vez.
<pre>else{ órdenes }</pre>	En caso de que no se cumpla una condición de tipo if previa, se reproducirá el diálogo de órdenes interior una vez.

A parte de estas estructuras condicionales, se cree necesario explicar también otra tipología condicional que ha sido la más utilizada a lo largo del programa ya que ha servido para ir analizando los paquetes de datos recibidos a partir de la aplicación móvil.

Ésta es la estructura *switchCase*.

Su estructura es la que se presenta a continuación:

```
Switch (variable){  
    Case 1:  
        Órdenes;  
    Break;  
    Case 2:  
        Órdenes;  
    Break;  
    ...  
    Default:  
        Órdenes;  
    Break;  
}
```

En función del valor de la variable, compara diferentes casos y si se cumple la igualdad ejecuta las órdenes correspondientes. En caso de que ningún case sea igual realiza las órdenes que hay dentro de default (por defecto).

9.3. Función cálculo movimiento

A continuación se presenta la parte del código que debe permitir el cálculo del posicionamiento de los servomotores con el fin de poder realizar movimientos del extremo del brazo articulado en función del eje de coordenadas de la base.

De este modo, siguiendo el modelo cinemático inverso calculado en el apartado 6.2 de este proyecto y aplicando las formulas del modelo en lenguaje de programación, se obtiene la siguiente función para el cálculo de movimientos:

```
void Calculo_XYZ()
{
    int R2=(Xf*Xf)+(Yf*Yf);

    float cosq3=((R2+(Zf*Zf)-(LONG_SL2*LONG_SL2)-(LONG_SL1*LONG_SL1))/(2*LONG_SL2*LONG_SL1));

    float sinq3=sqrt(1-cosq3*cosq3);
    q3=(atan(sinq3/cosq3))*180/PI;

    q2=(atan(Zf/-sqrt(Xf*Xf+Yf*Yf))-atan(LONG_SL1*sinq3/(LONG_SL2+LONG_SL1*cosq3)))*180/PI;
    float q21=(atan(Zf/+sqrt(Xf*Xf+Yf*Yf))-atan(LONG_SL1*sinq3/(LONG_SL2+LONG_SL1*cosq3)))*180/PI;

    q1=(atan(Yf/Xf))*180/PI;
}
```

9.4. Desarrollo del código

El desarrollo del código completo Arduino utilizado está disponible en el **Anexo B: Código Arduino** de de dicho proyecto.

10. Programación y diseño de la aplicación para dispositivo móvil

En este apartado se presentarán los medios que se usarán para el diseño de la aplicación Android para dispositivos móviles, juntamente con el aspecto final de dicha aplicación y el conjunto de pantallas que la forman.

10.1. Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones para Android. Entre sus características principales frente a otros entornos de desarrollo, se destacan las siguientes:

- Sistema de compilación flexible basado en Gradle.
- Emulador rápido con varias funciones.
- Entorno unificado en el que se puede realizar desarrollos para todos los dispositivos Android.
- Instant run, para aplicar cambios mientras la app está en ejecución.
- Integración de plantillas de código para ayudar a compilar funciones comunes de las apps e importar ejemplos de código.
- Gran cantidad de herramientas y frameworks de prueba.

10.1.1. Estructura del proyecto

Todos los proyectos en Android Studio contienen uno o más módulos con archivos de código fuente y archivos de recursos.

De forma predeterminada, Android Studio, muestra cada uno de los archivos de la aplicación que se está creando, de una manera organizada por módulos tal y como se muestra en la siguiente ilustración.

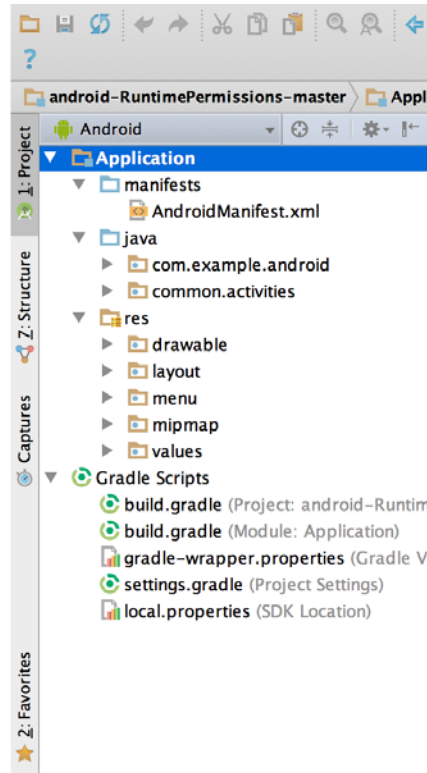


Ilustración 63. Estructura proyecto Android Studio

Como se puede observar, dentro de la carpeta Application, aparecen tres carpetas principales: manifests, java y res. En manifests se encuentra el archivo AndroidManifest.xml. La carpeta java contiene el conjunto de códigos fuente. Finalmente la carpeta res, contiene todos los recursos, diseños XML e imágenes.

10.1.2. Interfaz de usuario

La ventana principal de Android Studio presenta un aspecto tal y como el que se muestra a continuación (ilustración 64):

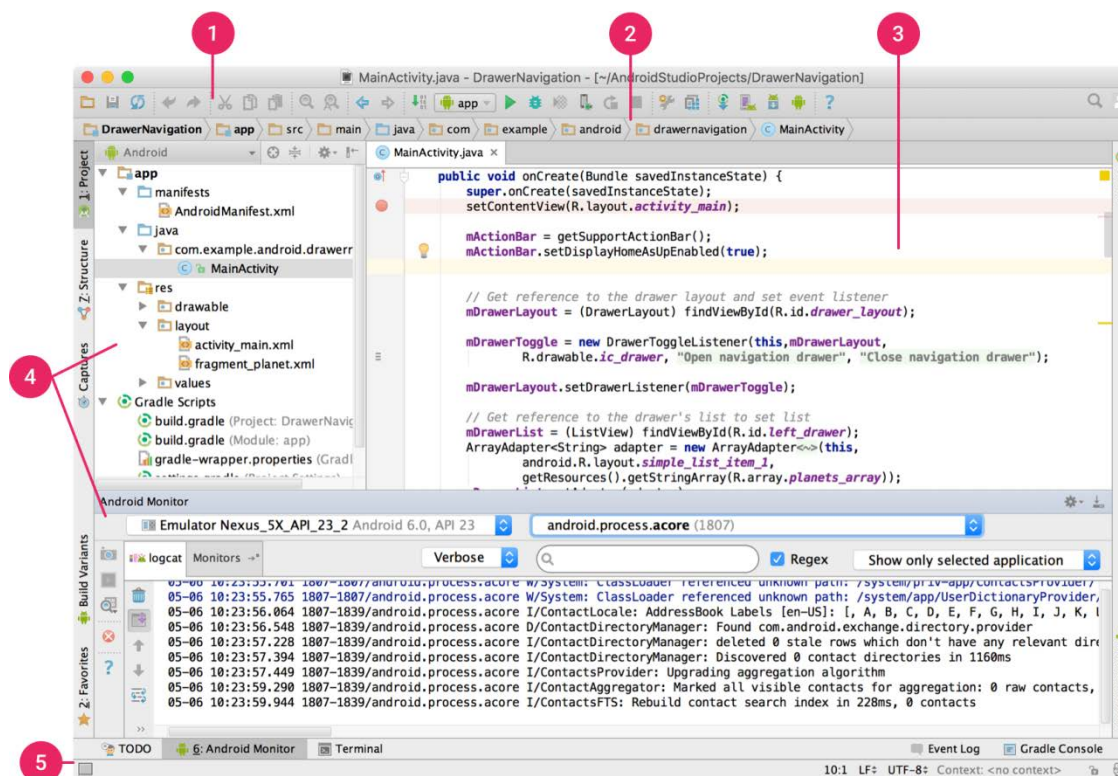


Ilustración 64. Interfaz de usuario Android Studio

Siendo cada zona de la pantalla:

- ① **Barra de herramientas:** permite realizar funciones tal y como la ejecución de la aplicación.
- ② **Barra de navegación:** indica en cada momento los archivos que están abiertos y que se pueden editar. Proporciona una vista más compacta de la navegación dentro de la carpeta app que se ha comentado anteriormente.
- ③ **Ventana del editor:** es la zona dónde el editor puede escribir el código de la aplicación o por otro lado crear el layout con la distribución de elementos que crearan la interface.
- ④ **Ventanas de herramientas:** te permite acceder a tareas específicas como la administración de proyectos, la búsqueda y los controles de versión.
- ⑤ **Barra de estado:** muestra el estado del proyecto, advertencias o mensajes.

10.1.3. Lenguaje de programación

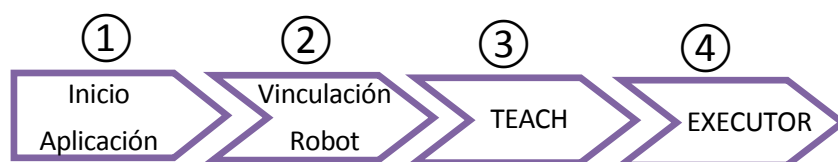
Como se ha comentado anteriormente, el lenguaje de programación que utiliza Android Studio es el lenguaje Java. Éste es el lenguaje más usado en el desarrollo de aplicaciones, ya que con este tipo de programación se pueden realizar aplicaciones de todo tipo. Además, Java permite la modularidad, es decir, la creación de rutinas individuales que pueden ser usadas por más de una aplicación y el desarrollo de aplicaciones bajo el esquema de cliente-servidor.

10.2. Diseño de la aplicación

La aplicación que permitirá el control del brazo robot estará formada por seis pantallas principales:

1. Menú principal.
2. Vinculación con el brazo robot.
 3. Instrucciones “teach”.
 - 3.1. Sistema de control JOINT.
 - 3.2. Sistema de control por coordenadas (X,Y,Z).
4. Instrucciones “executor”.
5. Instrucciones de uso y funcionamiento.

Ésta aplicación está diseñada para seguir una secuencia básica, tal y como se muestra en el siguiente diagrama.



La vinculación con el robot permitirá establecer la conexión entre aplicación y brazo articulado para empezar a dar las órdenes de movimiento deseadas.

En “TEACH” se permitirá realizar el movimiento del extremo del robot a la posición deseada moviendo el brazo a través de coordenadas X, Y, Z o a través del movimiento por articulaciones (JOINT).

10.2.1. Menú principal

El menú principal es la pantalla que permitirá navegar por la aplicación e intercambiar entre las diferentes pantallas que la forman. A parte de ser la pantalla inicial y la que aparecerá al iniciar la aplicación.

Su layout es el que muestra la siguiente ilustración:

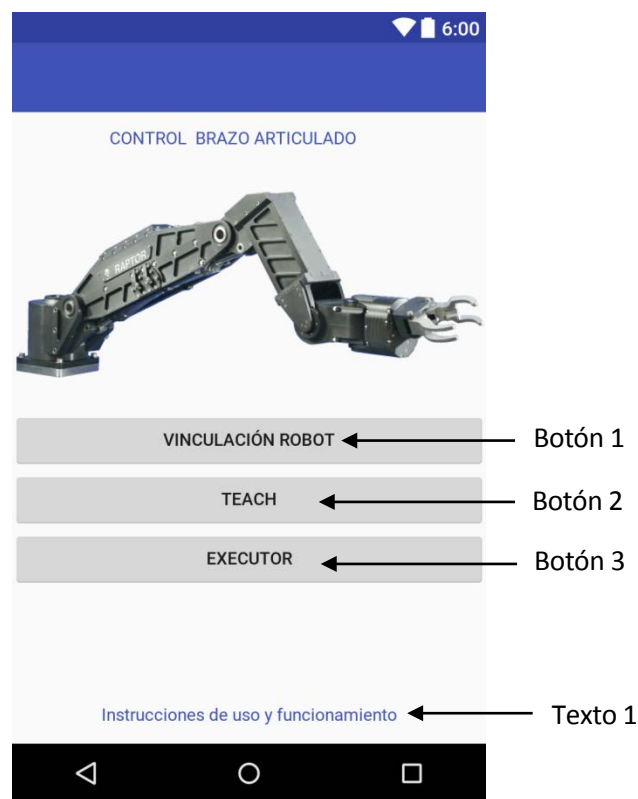


Ilustración 65. Pantalla menú aplicación

Como se puede observar está formada por tres botones, una imagen y dos textos. A continuación se explicara cuál es la función de cada componente que forma la pantalla.

Botón 1. “Vinculación robot”: se trata de un botón que al pulsarlo nos dará acceso a la pantalla dónde se podrá realizar la conexión Bluetooth con el brazo articulado.

Botón 2. “Teach”: al pulsarlo nos llevará a la pantalla dónde poder enseñar las posiciones al brazo articulado y guardarlas en un punto determinado para que posteriormente el robot sepa dónde están.

Botón 3. “Executor”: al pulsar este botón se conducirá a la pantalla dónde una vez guardadas las posiciones se le podrá indicar al robot la posición de la pinza en cada punto e iniciar un recorrido con la secuencia de puntos deseada.

Texto 1: "Instrucciones de uso y funcionamiento": se trata de un texto el cual se puede pulsar. Al hacerlo, éste conducirá a la pantalla dónde se encontrarán escritas las instrucciones básicas de funcionamiento del brazo robot.

Hay que tener en cuenta que mediante programación, en el momento en que se inicia la aplicación, hasta que no se ha vinculado el robot, no se puede acceder a ninguna de las funciones que permiten los botones "TEACH" y "EXECUTOR" ya que si no se ha establecido la comunicación entre ambos no serviría de nada tener acceso a estas pantallas.

10.2.2. Vinculación robot

Esta es la pantalla a la que se debe acceder al iniciar la aplicación. Su función, como su nombre indica, es iniciar la comunicación entre el dispositivo móvil y el brazo robot para poder realizar los movimientos.

A continuación se muestra su layout:

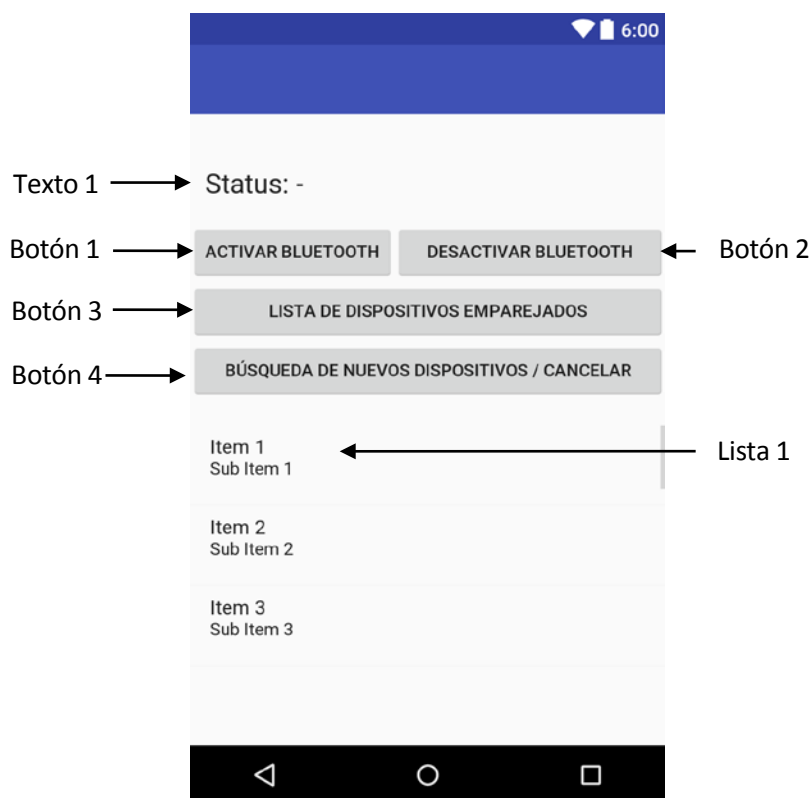


Ilustración 66. Pantalla vinculación aplicación

Texto 1. "Status": Se trata de un texto dónde se indica el estado del servicio Bluetooth del dispositivo móvil des del que se ejecuta la aplicación.

En caso de que el Bluetooth esté activado en este texto se mostrará: “Status: ACTIVADO”. Contrariamente, si el Bluetooth está desactivado, mostrará: “Status: DESACTIVADO”.

Botón 1. “ACTIVAR BLUETOOTH”: como su propio nombre indica, este botón permite la activación del Bluetooth del dispositivo. En caso de que al iniciar la aplicación este tipo de comunicación esté desactivada, este botón realizará una solicitud al usuario para que realmente autorice la activación.

Botón 2. “DEACTIVAR BLUETOOTH”: si el botón 1 permite activar el Bluetooth, el botón 2 permitirá desactivarlo. En principio no se debería desactivar, y por tanto no se debería usar este botón, pero el motivo por el que se ha implementado, es para dar la opción al usuario de desactivar la conexión en caso de que el dispositivo esté comunicado en el momento con otro dispositivo que no sea el brazo articulado.

Botón 3. “LISTA DE DISPOSITIVOS EMPAREJADOS”: al presionar este botón, se mostrarán en la lista 1 los dispositivos que el móvil ya tiene emparejados. Los dispositivos emparejados son aquellos con los que el teléfono móvil ya ha realizado un intercambio de datos previamente y que por lo tanto, ya lo tiene guardado como un dispositivo seguro. Al iniciar la conexión con los dispositivos emparejados, el teléfono no solicita al usuario ningún tipo de clave, ya que ésta ya se ha introducido en la primera conexión.

Botón 4. “BÚSQUEDA DE NUEVOS DISPOSITIVOS / CANCELAR”: al presionar este botón y con el Bluetooth del teléfono activado, el dispositivo empezará a buscar el conjunto de dispositivos con la conexión Bluetooth activada que están a su alcance. Todos los dispositivos que irán apareciendo se mostrarán en la lista 1 junto a los que ya tiene emparejados el móvil en caso de que se haya pulsado con anterioridad el botón 3. En caso de que una vez iniciada la búsqueda de nuevos dispositivos, se volviera a presionar este botón, se daría la búsqueda por finalizada.

Lista 1. Dispositivos disponibles: en esta lista que no tiene número de ítems máximo, si no que tendrá tantos ítems como dispositivos emparejados tenga el móvil y dispositivos disponibles dentro de la distancia de alcance de conexión Bluetooth que estén activados, aparecerán los nombres de los dispositivos junto a su dirección MAC. La dirección MAC de un dispositivo Bluetooth está compuesta por 12 números en el sistema hexadecimal agrupados de dos en dos y separado por “:”. Un ejemplo de dirección MAC sería: 01:23:45:67:89:ab.

10.2.3. TEACH

La función “TEACH” proviene del inglés enseñar. Esta pantalla lo que permite, es enseñar al robot unas posiciones concretas en el espacio para que las memorice. Lo que se consigue enseñando las posiciones al brazo articulado, es que posteriormente, sea capaz de moverse a la posición enseñada con completa autonomía y sin necesidad de que el usuario le de ninguna instrucción.

Los procedimientos más usados, en el sentido de facilitar las operaciones de enseñanza de los puntos, son dos: el movimiento por coordenadas y el movimiento por articulaciones. El usuario podrá escoger a partir de qué tipo de movimiento quiere llevar al robot al punto deseado para que memorice la posición de su extremo.

En el sistema JOINT el movimiento se realiza por articulaciones. Es decir, se selecciona la articulación que se quiere mover y a partir de aquí se produce un incremento o decremento de grados de posición.

En el sistema de coordenadas X,Y,Z el movimiento que realiza el brazo articulado es el desplazamiento en el eje X,Y o Z del sistema de coordenadas respecto la base del brazo articulado. Este tipo de movimiento, lógicamente, es más complejo que el movimiento por JOINT y requiere de una función intermedia que debe calcular el posicionamiento de cada articulación para poder conseguir moverse a través del eje deseado. Para esto será necesario el modelo de cinemática del brazo robot que se presentara más adelante.

A continuación se presentan los layout de las pantallas que nos permiten dar instrucciones tipo “teach” al robot. En este caso serán dos: JOINT e XYZ.

Layout JOINT:

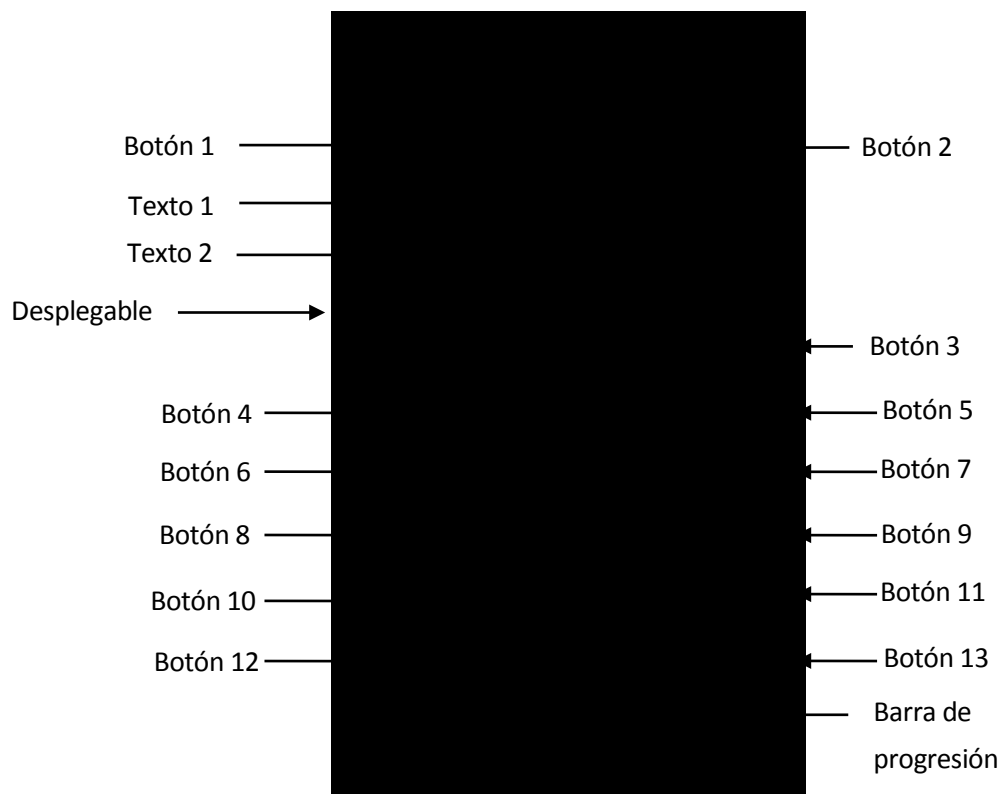


Ilustración 67. Pantalla JOINT aplicación

Botón 1. "XYZ": al pulsarlo muestra el layout de control por coordenadas X,Y,Z dentro de la función de la aplicación teach.

Botón 2. "JOINT": muestra, en este caso, el mismo layout para poder realizar el movimiento por articulaciones.

Texto 1. "J1, J2, J3, J4": muestra el estado en grados de la posición instantánea de las articulaciones.

Texto 2. "X, Y, Z": muestra la posición en el espacio del extremo del robot con el origen de coordenadas de la base del robot.

Desplegable: al pulsar la flecha aparece un desplegable dónde se da la opción de seleccionar entre cuatro puntos, que es el número máximo de puntos que podrá memorizar el robot.

Botón 3. "GUARDAR POSICIÓN": una vez seleccionado el punto dónde queremos guardar la posición, una vez se pulse este botón se realizara la memorización.

Botón 4/5: permite mover la posición de la articulación de la cintura hacia un lado u hacia otro.

Botón 6/7: permite mover la posición de la articulación del hombro hacia un lado u hacia otro.

Botón 8/9: permite mover la posición de la articulación del codo hacia un lado u hacia otro.

Botón 10/11: permite mover la posición de la articulación de la muñeca hacia un lado u hacia otro.

Botón 12/13: permite mover la posición de la mano hacia un lado u hacia otro.

Barra de progresión: la barra de progresión tiene tres posiciones y al modificar su estado permitirá hacer un movimiento más corto o más largo a decisión del usuario en función de si el movimiento que quiere realizar dista mucho de la posición actual.

Layout X, Y, Z:

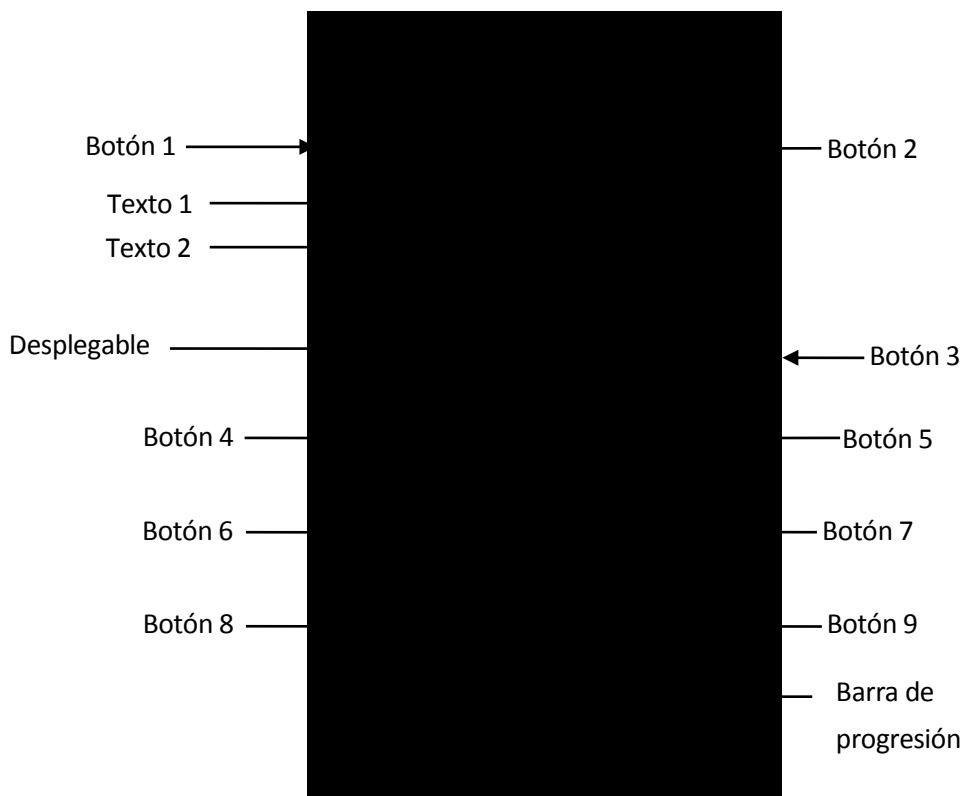


Ilustración 68. Pantalla X,Y,Z aplicación

Como podemos observar en esta captura de pantalla la mitad superior corresponde al mismo layout que las instrucciones JOINT. En este caso, lo único que cambia es que los botones 4,5,6,7,8 y 9 permiten mover el extremo del robot de manera positiva o negativa en cada uno de los ejes de coordenadas. La barra de progresión, a su vez, tiene la misma función que anteriormente.

10.2.4. EXECUTOR

Una vez se le han enseñado al brazo robot las posiciones deseadas. En este caso cuatro. Es momento de ir a la pantalla executor (ejecutor en castellano). En esta pantalla se podrá elegir la posición que queremos que tenga la pinza en cada uno de los puntos que anteriormente se han memorizado e iniciar un recorrido con los puntos previamente guardados.

A continuación se muestra el layout de la pantalla executor:

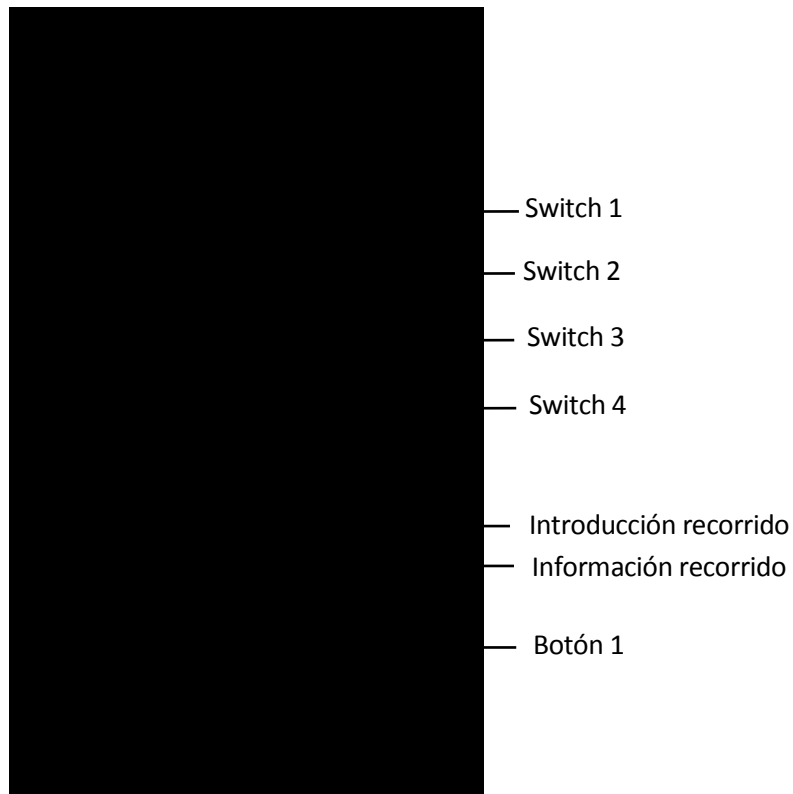


Ilustración 69. Pantalla ejecutor aplicación

Switch 1: permite escoger la posición de la pinza en el punto uno memorizado.

Switch 2: permite escoger la posición de la pinza en el punto dos memorizado.

Switch 3: permite escoger la posición de la pinza en el punto tres memorizado.

Switch 4: permite escoger la posición de la pinza en el punto cuatro memorizado.

Introducción recorrido: se trata de una entrada de datos dónde por predeterminación aparece “Ejemplo: 1, 2, 3, 4”. Al pulsar sobre este texto, permite ser modificado. Como se indica en las instrucciones de uso y funcionamiento del brazo robot y aplicación, en esta sección el usuario debe introducir un código formado siempre por cuatro dígitos del número cero al cuatro y separados por comas.

Es decir el mínimo recorrido que introduce el usuario debe ser de un punto y el máximo de cuatro. En caso de que el usuario decida hacer un recorrido de menos de cuatro puntos y más de uno deberá posicionar un cero en las posiciones que no vaya usar.

A continuación se muestran ejemplos de recorrido válido que se podría introducir para aclarar el sistema de introducción:

Recorrido de cuatro puntos diferentes: 2, 4, 1,3.

Recorrido de cuatro puntos con puntos iguales: 2, 1, 2,1.

Recorrido de tres puntos: 3, 1, 2, 0.

Recorrido de dos puntos: 4, 3, 0, 0.

Recorrido de un punto: 2, 0, 0, 0.

Una vez se inicie el recorrido en cuanto el extremo del brazo articulado llegue a la posición deseada abrirá o cerrará la pinza en función del estado del Switch correspondiente. Hay que indicar que la pinza siempre inicia el recorrido con la pinza abierta.

Información de recorrido: se trata de un texto que inicialmente de manera predeterminada mostrará la palabra "RECORRIDO". A partir de aquí, una vez se haya introducido el recorrido y se pulse el botón de iniciar recorrido, en función de los parámetros introducidos por el usuario se mostrará diferentes mensajes:

- "RECORRIDO VÁLIDO": el usuario ha introducido un recorrido válido y por lo tanto cuando se pulse el botón se podrá iniciar sin problema el recorrido deseado.
(Ej: 1, 2, 3, 4)
- "SOBRAN PUNTOS": mostrará este mensaje cuando el usuario haya introducido más de cuatro puntos en el recorrido.
(Ej: 2, 3, 1 ,4, 2)
- "FALTAN PUNTOS": este mensaje aparecerá cuando el usuario introduzca menos de cuatro puntos.
(Ej: 2,3)
- "DATO VACÍO": aparecerá cuando el usuario se deje por introducir un punto de los cuatro que forman el recorrido.
(Ej: 2,3,,1)
- "LETRAS NO VÁLIDAS": cuando el usuario en vez de un numero del uno al cuatro introduzca una letra en un punto del recorrido.
(Ej: a, 4, 1, 4)
- "RECORRIDO NO VÁLIDO": aparecerá este mensaje cuando se introduzca un número superior a cuatro. Es decir, un punto memorizado no existente.
(Ej: 1, 2, 3, 5)

Botón 1. “INICIAR RECORRIDO”: por último, al pulsar este botón, y siendo el texto de información recorrido “RECORRIDO VÁLIDO”, el brazo robot iniciará el conjunto de movimientos.

10.3. Simulaciones

Para realizar las simulaciones se utilizará el emulador GenyMotion para las operaciones más sencillas y un dispositivo móvil Android real para las operaciones más complejas como el envío de datos a través de Bluetooth.

10.3.1. Genymotion

Genymotion es un emulador para la ejecución de aplicaciones Android completamente gratuito. Te permite la creación de teléfonos móviles virtuales (con limitaciones) de todos los modelos de marcas de teléfonos móviles. Entre la selección de dispositivos no se limita sólo a teléfonos o smartphones sino que también te deja seleccionar entre tablets y smartwatches.

Para instalar la aplicación en el emulador virtual sólo hace falta seleccionar el Smartphone deseado e iniciarlo des de la misma aplicación. En este caso, las simulaciones se han realizado con el dispositivo Samsung Galaxy S3.

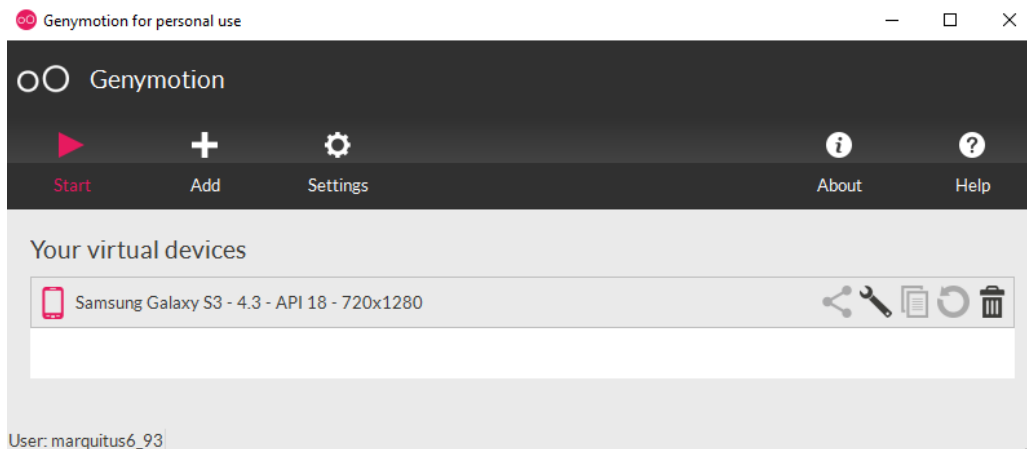
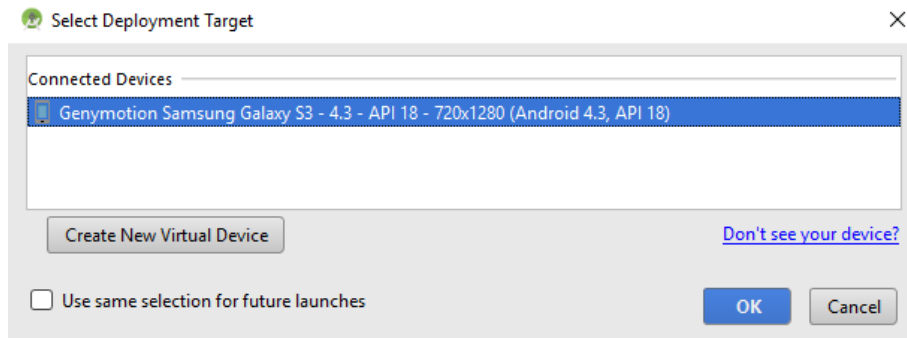


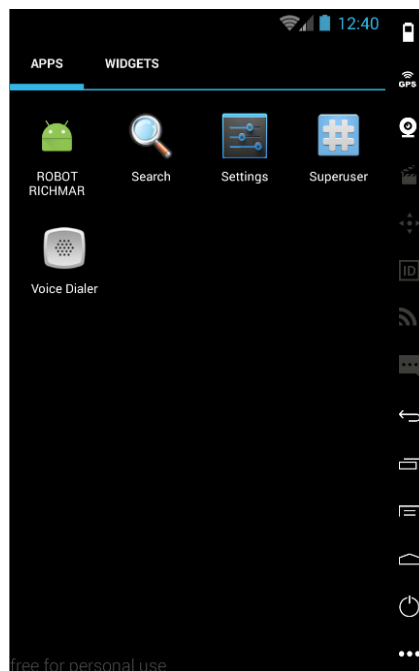
Ilustración 70. Simulador Genymotion (I)

Una vez inicializado el dispositivo virtual, sólo es necesario hacer correr la aplicación des de Android Studio en este dispositivo.



Il·lustració 71. Simulador Genymotion (II)

Una vez cargada la aplicaci3n al dispositivo virtual, tan s3lo hay que buscarla en el dispositivo y ejecutarla.



Il·lustraci3n 72. Simulador Genymotion (III)

Como se puede observar, en el men3 de aplicaciones del dispositivo, aparece la aplicaci3n con el s3mbolo de Android y el nombre de ROBOR RICHMAR que es el nombre que se le ha dado a la aplicaci3n.

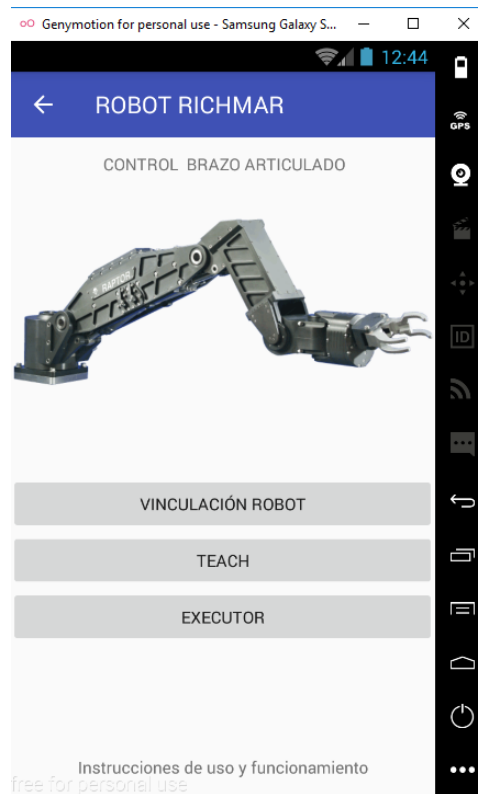


Ilustración 73. Simulador Genymotion (IV)

Aunque este emulador ha sido muy útil para operaciones básicas de la aplicación, en cuanto la complejidad de la aplicación ha ido aumentando ha sido necesario pasar a utilizar un dispositivo móvil o Smartphone real para llevar a cabo las pruebas.

10.3.2. Smartphone

El Smartphone usado para realizar todas las simulaciones ha sido un BQ Aquaris E4.5. El procedimiento para cargar la aplicación al dispositivo es el mismo que se ha llevado a cabo con el emulador Genymotion, aunque en este caso, ha sido necesario habilitar la opción de “programador” dentro del sistema operativo Android del dispositivo.

A continuación se mostrará la secuencia básica de la simulación de la aplicación móvil siguiendo los pasos básicos descritos anteriormente y que sería la secuencia básica que debería seguir un usuario.

1.

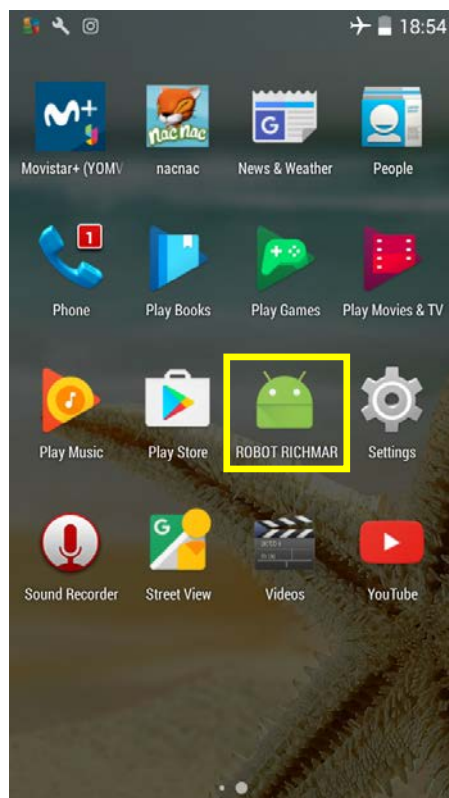


Ilustración 74. Ejecución real aplicación 1

2.



Ilustración 75. Ejecución real aplicación 2

Cómo se puede observar en la pantalla 1, al entrar al menú de aplicaciones, el dispositivo ya muestra la aplicación que permitirá enlazar e iniciar la comunicación entre el brazo robot a la vez de permitir dar las ordenes de movimientos requeridas.

En la pantalla 2, se muestra cómo al iniciar la aplicación tan sólo se puede pulsar el botón de vinculación con el robot. En caso de pulsar cualquiera de los otros dos botones (teach o executor) se mostrará una advertencia como la que aparece en la que se indica que es necesario antes de todo emparejar el robot.

3.



4.

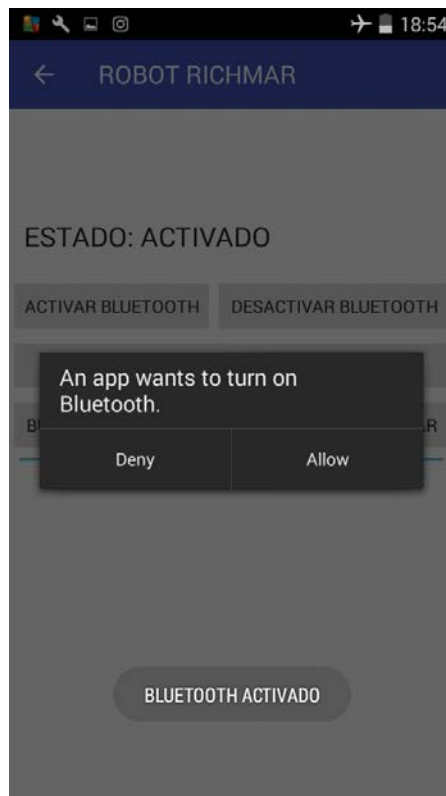


Ilustración 76. Ejecución real aplicación 3

Ilustración 77. Ejecución real aplicación 4

Una vez el usuario ya pulsa el botón correspondiente (vinculación robot) se accede a la pantalla donde se deberá activar el Bluetooth (en caso de que no esté activado). Una vez el teléfono ya tiene este tipo de comunicación activada, tan solo será necesario mostrar los dispositivos emparejados (en caso de que se haya emparejado el robot con anterioridad) o por el contrario iniciar la búsqueda de dispositivos.

Cuando se muestre el módulo Bluetooth del robot, sólo será necesario presionarlo de entre la lista del resto de dispositivos y la aplicación dará por iniciada la comunicación con el módulo conectado a Arduino, es decir, con el brazo articulado.

Una vez establecida la conexión la aplicación mostrará la pantalla inicial dónde será momento de acceder al botón teach para enseñarle las posiciones deseadas al robot.

Una vez pulsado el botón teach, cómo pantalla predeterminada aparecerá la pantalla de movimientos JOINT.

Una vez dentro se podrá intercambiar cuantas veces se desee entre el modo de enseñanza de posiciones JOINT e X, Y, Z.

5.

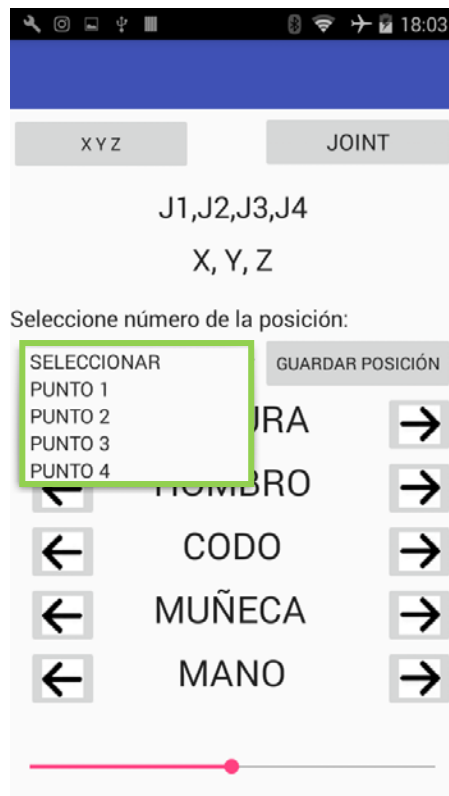


Ilustración 78. Ejecución real aplicación 5

6.

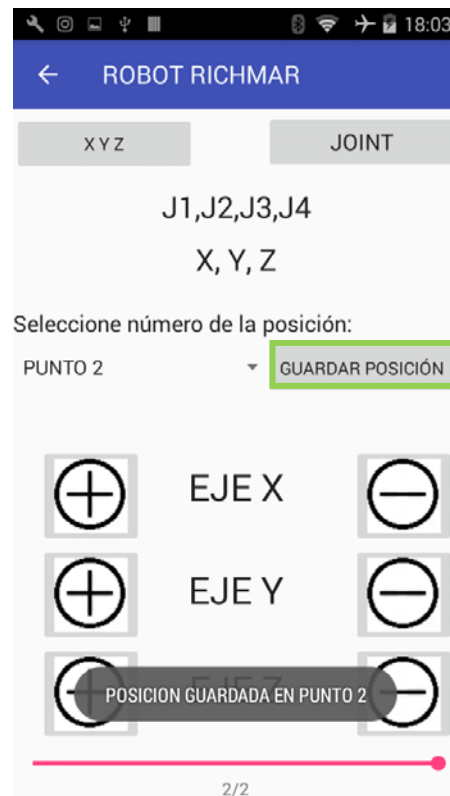


Ilustración 79. Ejecución real aplicación 6

En la pantalla 6 se muestra como se puede seleccionar en el desplegable el punto en el que se quiere guardar la posición. Sólo será necesario llevar el extremo a una posición con la longitud de movimiento deseada, seleccionar el punto y guardarlo.

Una vez se ha seleccionado el punto (pantalla 7) al presionar el botón guardar posición este punto se guardará, apareciendo en pantalla el aviso de que la posición ha sido guardada.

Una vez se hayan guardado las posiciones, como se puede observar en todas la pantallas presentadas, en la parte superior izquierda, aparece una flecha hacia la izquierda. Al presionarla, esté donde esté, la aplicación conducirá a la pantalla inicial. Entonces pues, una vez guardados todos los puntos, será necesario pulsar esta flecha para volver a la página principal y entrar a la pantalla executor.

Una vez dentro de executor, sólo será necesario introducir las posiciones de las pinzas en los puntos guardados, marcar el recorrido que se quiere llevar a cabo y pulsar el botón de iniciar recorrido y a partir de aquí, y siempre y cuando el recorrido esté bien introducido, el robot iniciará los movimientos.

7.

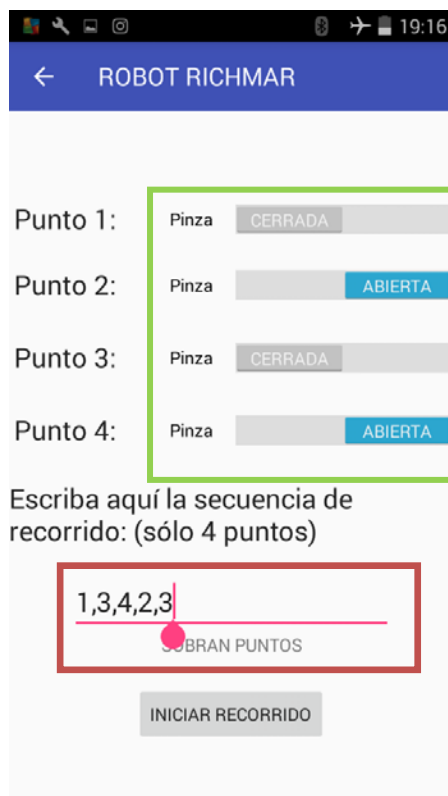


Ilustración 80. Ejecución real aplicación 7

8.

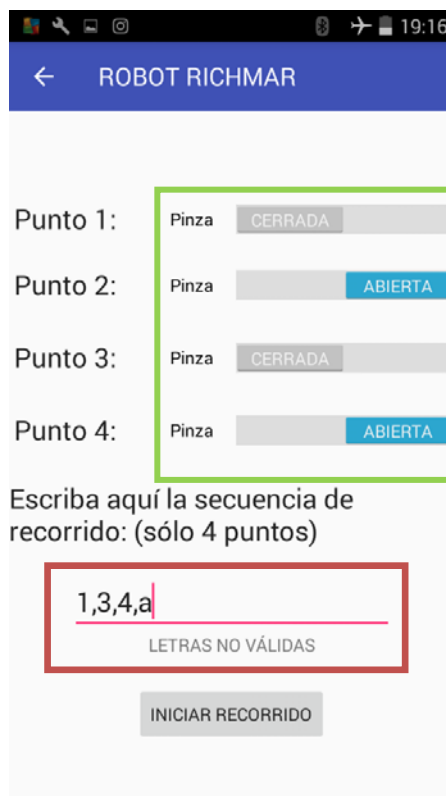


Ilustración 81. Ejecución real aplicación 8

En las pantallas 8 y 9 cómo se ha explicado anteriormente, al introducir un recorrido que no sigue las pautas, el texto que aparece debajo de la introducción del conjunto de puntos cambia mostrando el error que está cometiendo el usuario a la hora de introducir los datos.

En la pantalla 8, como se puede apreciar, el usuario ha introducido más de 4 puntos separados por comas. En el caso de la pantalla 9, el usuario ha introducido una letra, en vez de un punto guardado y por lo tanto se le está marcando el error de que las letras no son válidas.

Siguiendo con la ejemplificación de más errores, las pantallas 10 y 11 nos muestran errores que podrían cometer los usuarios al entrar el recorrido.

9.

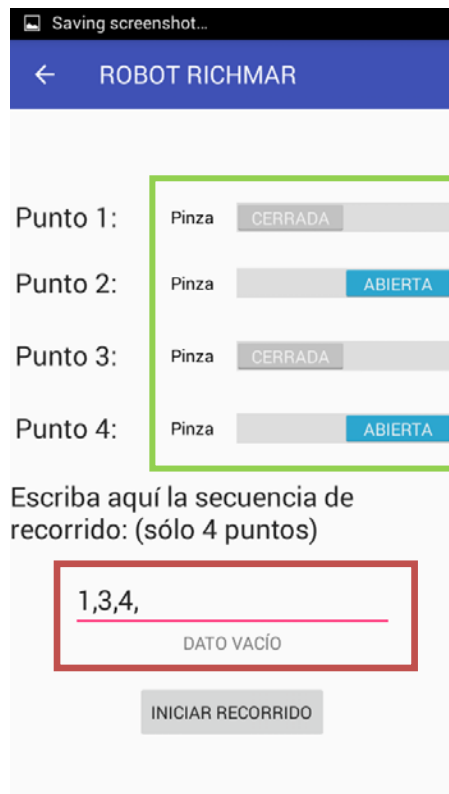


Ilustración 82. Ejecución real aplicación 9

10.

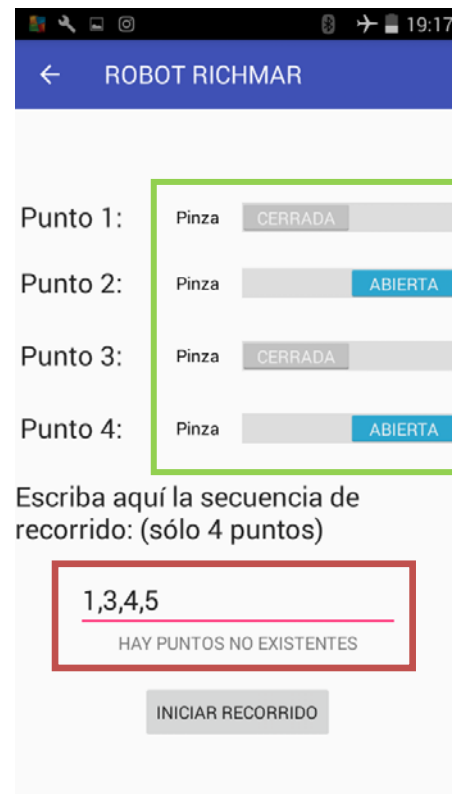


Ilustración 83. Ejecución real aplicación 10

En el caso de la pantalla 10, se indica que hay un dato vacío ya que en caso de que el usuario sólo quisiera ir del punto 1 al punto 3 y del punto 3 al punto 4 debería haber introducido el recorrido 1, 3, 4, 0.

Por otro lado, en el caso de la pantalla 11, el usuario ha indicado que quiere desplazar el brazo articulado al punto 5, pero se debe tener en cuenta que el punto 5 no puede haber estado enseñado al robot ya que el número máximo de puntos que se le pueden enseñar son 4.

Ya por último, cuando el usuario introduce un recorrido válido, como el que se muestra en la pantalla 12, el robot inicia el recorrido marcado.

11.

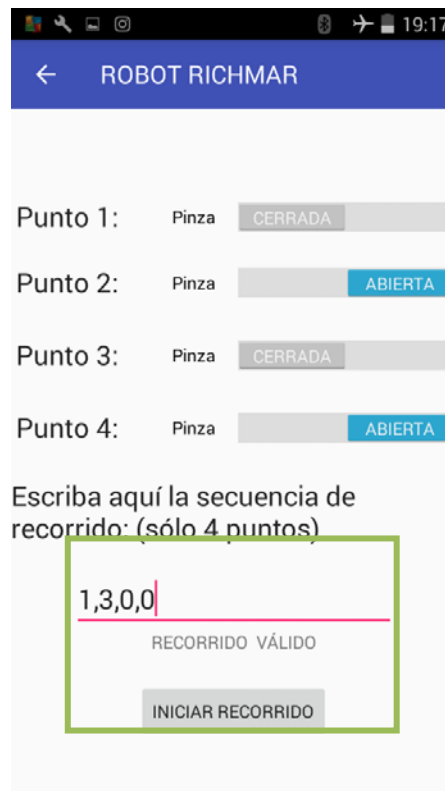


Ilustración 84. Ejecución real aplicación 11

Es lógico pensar qué hace el brazo una vez finalizado el recorrido, o qué pasa con su posición y la aplicación. Pues bien, una vez el brazo ha finalizado el recorrido, se queda en la posición del último punto al que se ha desplazado. A partir de aquí si se desea, se puede volver a iniciar el mismo recorrido o iniciar el mismo recorrido con diferentes posiciones de la pinza o incluso realizar el recorrido con otra secuencia. Si lo que se desea es enseñar al robot otras posiciones sólo será necesario pulsar el botón de retroceder que se ha introducido anteriormente y acceder a la pantalla de teach. Cuando se guarden nuevos puntos, éstos “machacarán” los que había anteriormente y se sobrescribirán. En caso de que haya algún punto que no se enseñe de nuevo seguirá con la posición que se le había dado en la última ocasión.

10.4. Programa Android Studio

El programa utilizado para la aplicación del dispositivo móvil se encuentra adjunto en el **Anexo D: Código Android Studio** de este proyecto.

11. Construcción del prototipo

11.1. Elementos del brazo articulado

11.1.1. Base

Para la base del brazo articulado se ha usado un freno de disco de un automóvil ya que es un elemento pesado y que ofrece la posibilidad de poner un soporte en su interior que fijará el servomotor de la cintura. Solo ha sido necesario realizar los agujeros necesarios para el anclaje de la parte fija de la articulación.

A continuación se muestran los elementos que permiten el movimiento rotacional de la cintura.

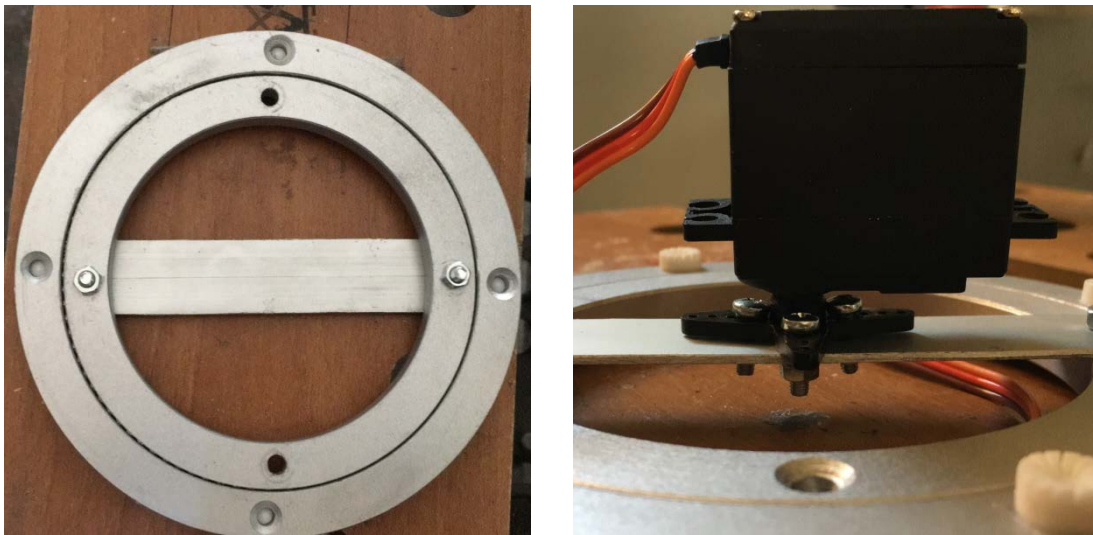


Ilustración 85. Construcción de la base(I)



Ilustración 86. Construcción de la base (II)

Como se puede observar se ha añadido una placa de aluminio al elemento giratorio que nos permite anclar el servomotor a la parte interna de la articulación. Observando la imagen anterior, el círculo exterior de la articulación se debe fijar al freno de disco y la parte interior a la base del robot.

Los servomotores que se sitúan en la base han estado preparados de la siguiente manera para tener los engranajes mediante los cuales moverán la correa y transmitirán el movimiento.

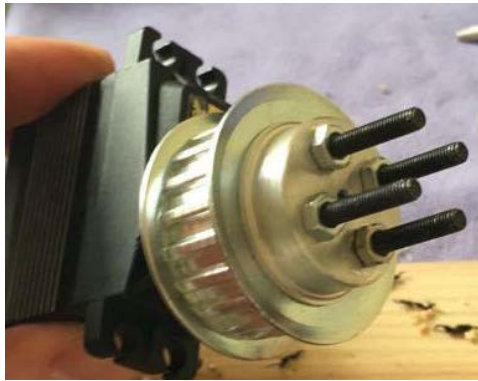


Ilustración 87. Preparación servomotores

En el caso del servomotor para el movimiento de la mano, ha sido necesario hacer un soporte que lo fijará a la base tal y como se muestra en la imagen de la derecha.

Seguidamente se presenta la base a la que estarán localizados los servomotores del hombro, codo y muñeca.

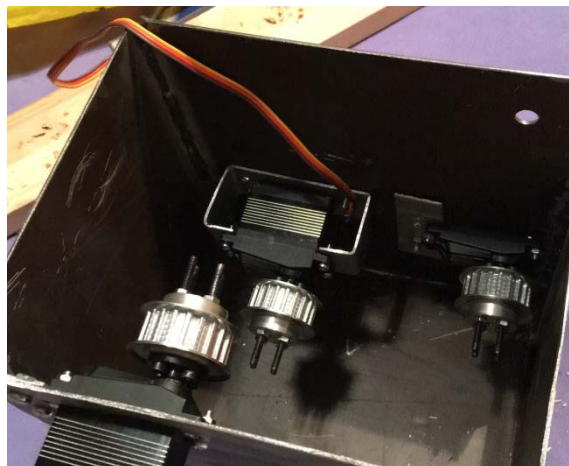


Ilustración 88. Construcción de la estructura de la base

La base ha sido soldada y en ella se han realizado los agujeros necesarios para poder anclar tanto los servomotores, los sensores, ejes y otros elementos de carácter mecánico.

Por último el aspecto final que presenta el interior de la base:

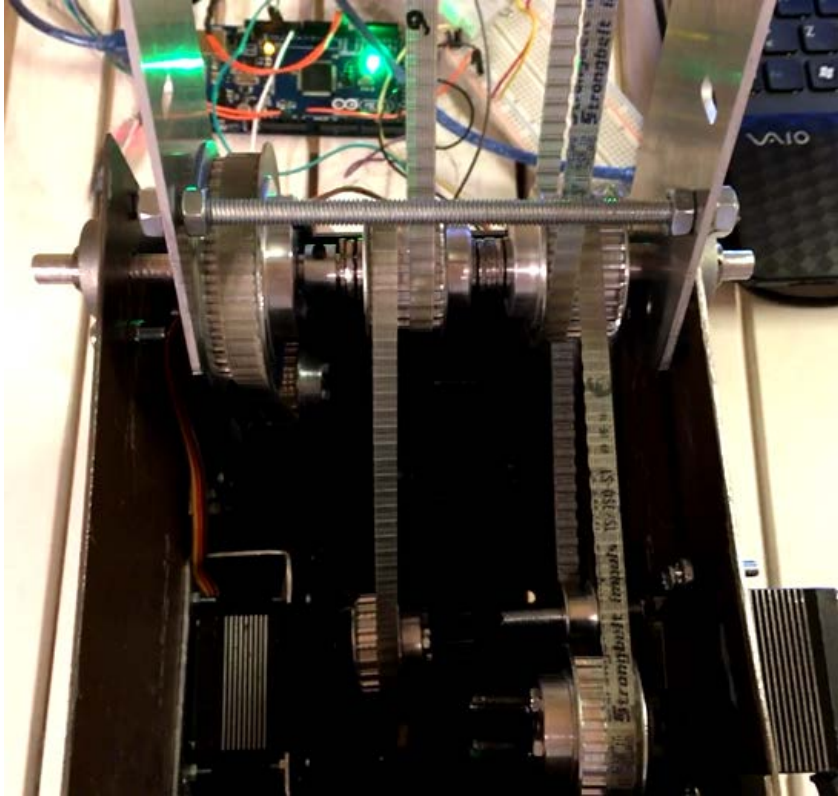


Ilustración 89. Base del prototipo

11.1.2. Eslabones

A continuación se muestra el aspecto del brazo formado por los dos eslabones, y las articulaciones de hombro codo y mano.



Ilustración 90. Construcción eslabones del prototipo

En esta ilustración podemos observar que los eslabones están formados por cuatro placas. Dos para el primero y dos para el segundo idénticas entre ellas.

En este caso, para conseguir máxima exactitud en todos los agujeros tanto para tensores como para ejes etc. los cortes han sido realizados con láser a través de una empresa del sector.

Para tener más detalle, se adjuntan imágenes de la articulación del hombro, codo y muñeca:

Hombro:

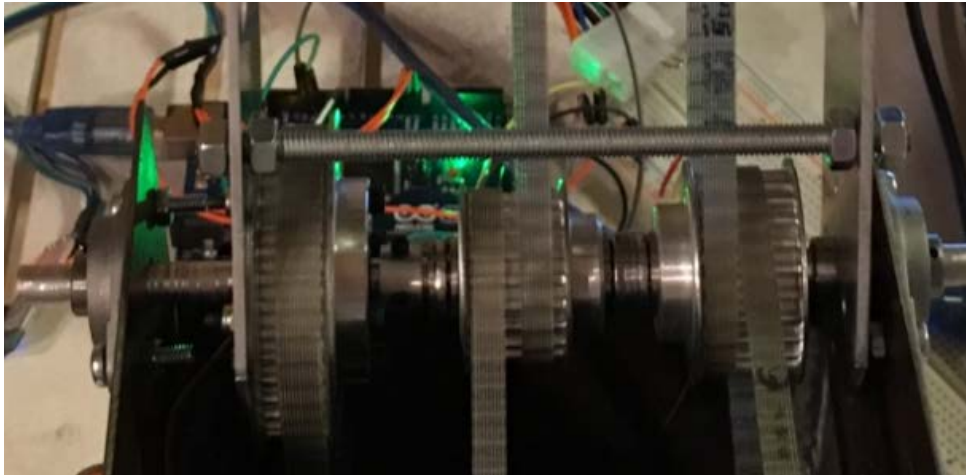


Ilustración 91. Articulación del hombro del prototipo

Codo:

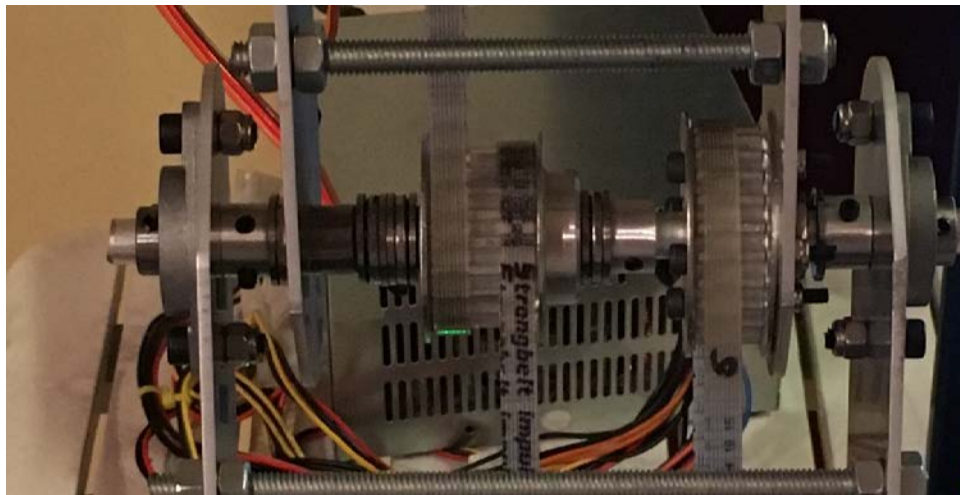


Ilustración 92. Articulación del codo del prototipo

Muñeca:

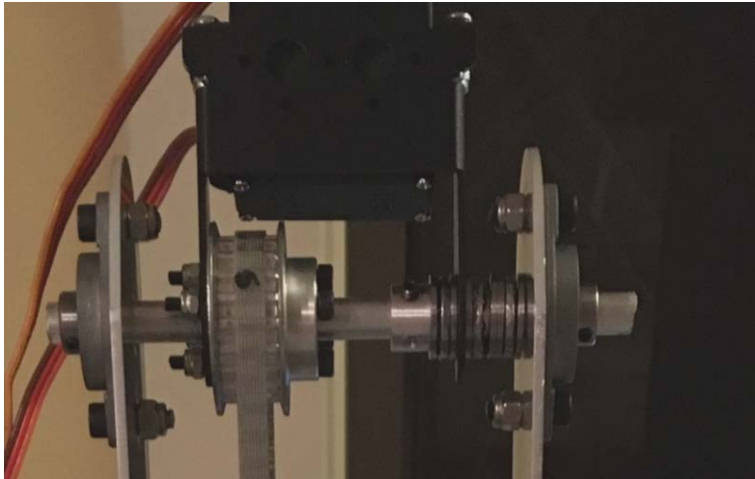


Ilustración 93. Articulación muñeca del prototipo

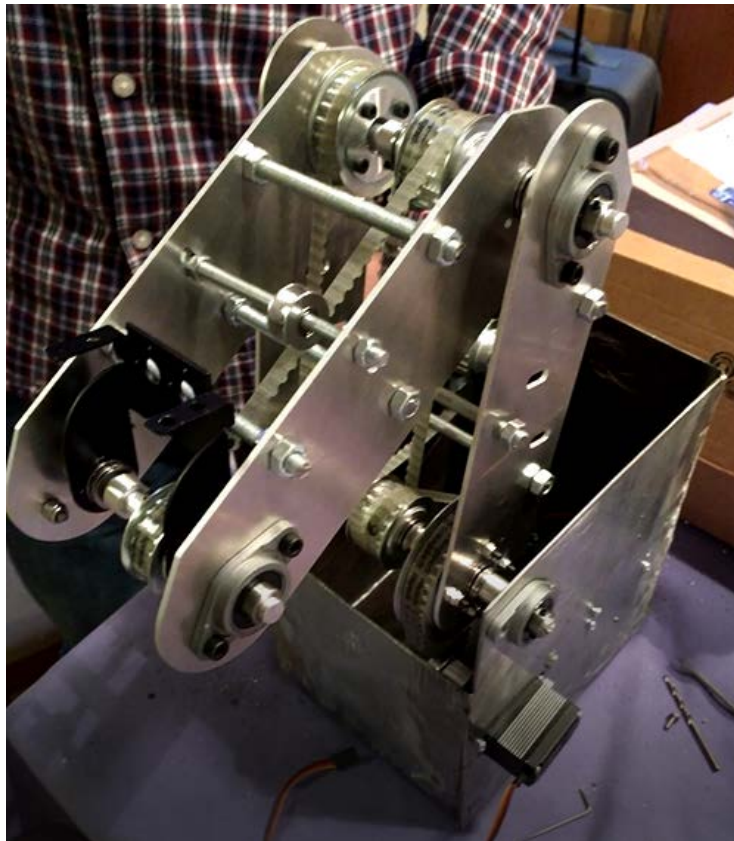


Ilustración 94. Aspecto del brazo articulado

Pinza

La pinza del brazo robot articulado es un elemento el movimiento del cual se realiza a partir de dos servomotores. Un servomotor nos servirá para mover la muñeca del brazo y el otro para abrir y cerrar dicha pinza.

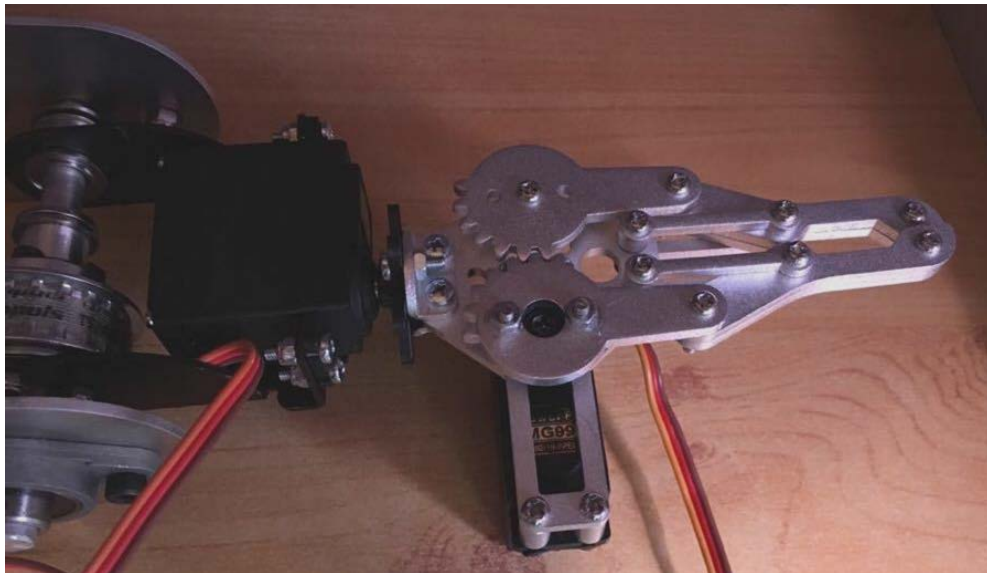


Ilustración 95. Montaje pinza



Ilustración 96. Montaje estructura del brazo

11.2. Brazo articulado

Por último, se presenta el aspecto final del brazo articulado.

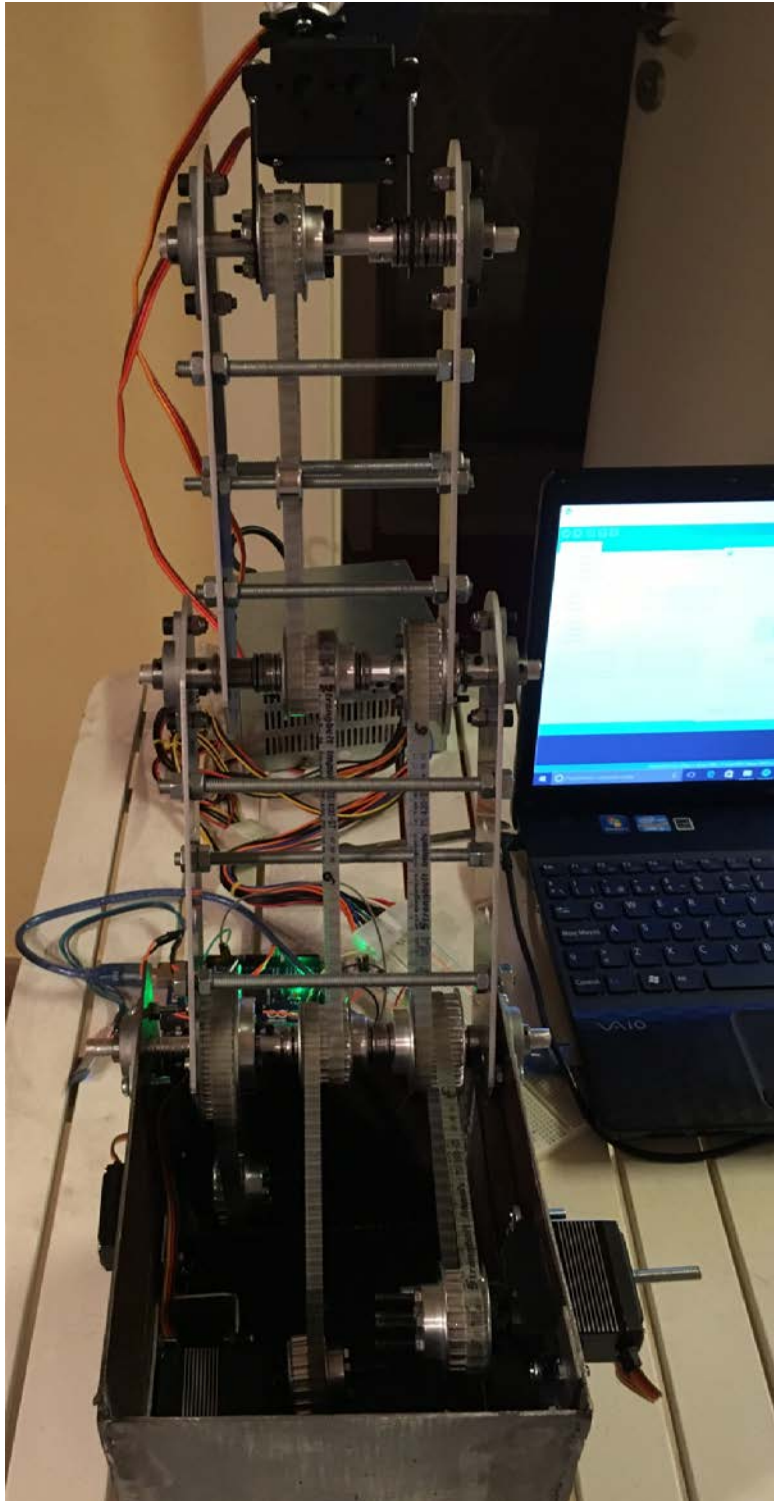
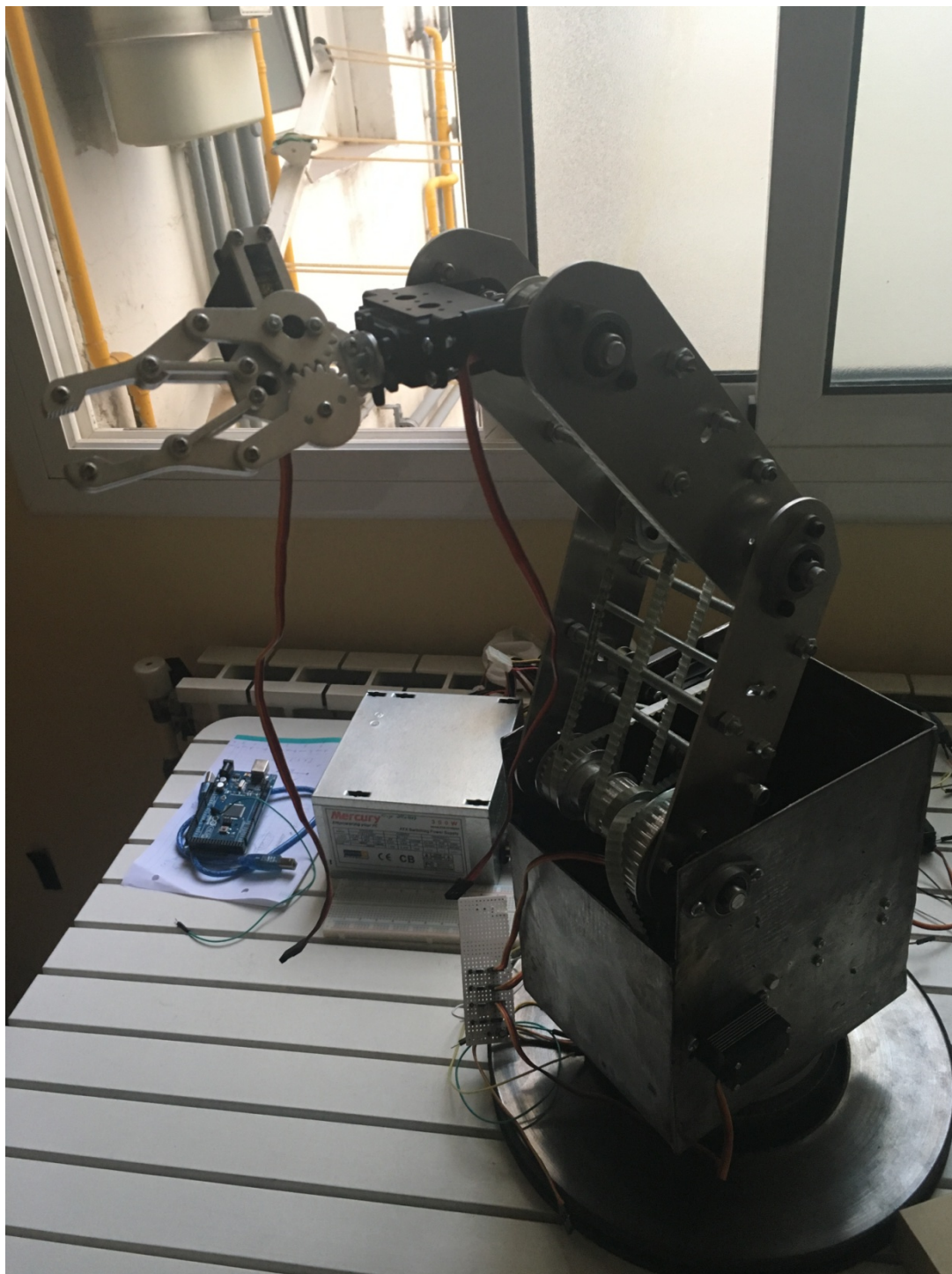


Ilustración 97. Aspecto final del brazo articulado (I)



Il·lustració 98. Aspecto final del brazo articulado (II)



12. Análisis del impacto ambiental

El impacto que puede tener un proyecto de estas dimensiones al medio ambiente puede ser estudiado desde el punto de vista del impacto que puede provocar el material del cual está formada su estructura.

El material protagonista de la estructura del brazo articulado del prototipo construido es el aluminio. Se trata de uno de los materiales más rentables de la industria ya que se aprovecha prácticamente en su totalidad.

Su proceso de reciclaje empieza con su recolección y traslado a la planta de reciclaje donde es seleccionado, limpiado y aplastado formando grandes bloques. Una vez creados estos bloques, se funden y se crean nuevas láminas que estarán listas para un nuevo uso.

No obstante, el aluminio, un material muy abundante en el planeta, requiere de un proceso de extracción de altos costes, especialmente energéticos. Es por esto, que reciclándolo se contribuye a reducir estos costes. Aparte, cabe destacar que deben pasar varios procesos de reciclaje para que el material pierda sus características.





14. Estudio económico

14.1. Costes de desarrollo y prototipado

En este apartado se presentan los costes relacionados con el coste de ingeniería y materiales para la construcción del prototipo.

Por un lado el coste de ingeniería es el dinero invertido en personal para el desarrollo del prototipo y por otro lado, el coste de material, es el precio del conjunto de materiales físicos necesarios para la construcción del brazo articulado.

14.1.1. Costes de ingeniería

El personal involucrado en este proyecto han sido dos ingenieros técnicos de categoría junior. A continuación se presenta el tiempo dedicado a cada una de las tareas que se han

Tabla 23. Costes de ingeniería

COSTES DE INGENIERÍA	NÚMERO DE INGENIEROS INVOLUCRADOS EN LA TAREA	TIEMPO (H)
Propuesta/idea de proyecto	2	20
Estado del arte	1	20
Definición del objetivo y alcance	2	20
Especificaciones	2	30
Planificación	2	30
Diseño mecánico	1	250
Selección de componentes y compra	2	50
Diseño electrónico	2	300
Diseño de la aplicación móvil	2	400
Presupuesto	1	20
Impacto ambiental	1	20
vConstrucción del prototipo	2	400
Realización de planos	2	40
Realización de informes y anexos	2	100
TOTAL HORAS:		1.700

Suponiendo que un ingeniero júnior puede cobrar 20 € la hora se obtiene un coste total de **34.000€**.

14.1.2. Costes de materiales

A continuación se muestra el coste de todos los productos necesarios para la fabricación del prototipo.

Tabla 24. Costes de materiales

COSTES DE FABRICACIÓN	COSTE UNITARIO (€)	CANTIDAD (UDS)	COSTE TOTAL (€)
MATERIAL ESTRUCTURAL			
Pieza aluminio brazo hombro	26	2	52
Pieza aluminio brazo codo	24	2	48
Pinza	18	1	18
Pieza pref. Servomotores	16	1	16
Ejes aluminio 8 mm (1m)	1.95	1	1,95
Eje roscado 6 mm (1m)	2.1	2	4,20
Eje roscado 5 mm (1m)	2.05	1	4,10
Base giratoria	14	1	14
Material chatarrería(disco)	20	1	20
Placa base	30	1	30
FIJACIONES			
Soportes KFL08 (pack 10)	28	1	28
Anillos de fijación 8mm (pack 8)	14	1	14
Rodamiento axial F8-16	1.55	20	31
Rodamiento radial HK0810	1.7	20	34
Tornillería varia	4	3	12
ARTICULACIONES			
Polea dentada T5 36 10mm	13	1	13
Polea dentada T5 20 16mm	9	3	27
Polea dentada T5 20 10mm	9	4	36
Polea dentada T5 24 10mm	11	1	11
Polea dentada T5 24 16mm	11	1	11
Poleas tensoras	9	2	18
Correa dentada T5-350 6mm	7.4	1	7,40
Correa dentada T5-400 6mm	7.48	3	22,44
Correa dentada T5-420 6mm	7.48	1	7,48
Correa dentada T5-350 8mm	9.44	1	9,44
ELECTRÓNICA			
Servomotor Tower Pro MG995	14,12	2	28,24
Servomotor Tower Pro MG958	14,95	3	44,85
Servomotor Tower Pro MG959	32	1	22,48
Placa Arduino Mega 2560	42,98	1	42,98
Fuente alimentación	20	1	20
Teléfono móvil BQ Aquaris E4.5	150	1	150
Diodos	0,04	6	0,24
Módulo Bluetooth HC06	5,49	1	5,49
TOTAL			813,81€

14.1.3. Coste total del desarrollo y prototipado

Por último se tomará como coste total del proyecto, el sumatorio de los dos costes presentados.

Tabla 25. Coste total del desarrollo y prototipado

Costes de ingeniería	34.000 €
Costes de fabricación	813,81 €
COSTE TOTAL	34.813,81€

14.2. Coste de venta al público

A continuación se realiza un estudio de precio de venta al público del prototipo. Para su realización será necesario dividir los costes en los siguientes apartados:

- Coste de materiales.
- Coste de montaje.
- Coste de amortización.
- Ganancias.
- Precio final de venta.

Se considerará una fabricación inicial de 50 unidades ya que se considera que es el número estimado de productos que se podrán vender en el período de los siguientes cuatro años.

14.2.1. Coste de materiales

En este apartado se considerará que el coste unitario del material se reduce un **10%** del coste de una unidad debido a la compra de una cantidad más elevada.

Tabla 26. Costes de materiales

COSTES DE FABRICACIÓN	COSTE UNITARIO (€)	COSTE UNITARIO CON DTO. (€)	CANTIDAD (UDS)	COSTE TOTAL (€)
MATERIAL ESTRUCTURAL				
Pieza aluminio brazo hombro	26	23,4	100	2340
Pieza aluminio brazo codo	24	21,6	100	2160
Pinza	18	16,2	50	810
Pieza pref. Servomotores	16	14,4	50	720
Ejes aluminio 8 mm (1m)	1,95	1,755	50	87,75
Eje roscado 6 mm (1m)	2,1	1,89	100	189

Eje roscado 5 mm (1m)	2,05	1,845	50	92,25
Base giratoria	14	12,6	50	630
Material chatarrería(disco)	20	18	50	900
Placa base	30	27	50	1350
FIJACIONES				
Soportes KFL08 (pack 10)	28	25,2	50	1260
Anillos de fijación 8mm (pack 8)	14	12,6	50	630
Rodamiento axial F8-16	1,55	1,395	1000	1395
Rodamiento radial HK0810	1,7	1,53	1000	1530
Tornillería varia	4	3,6	150	540
ARTICULACIONES				
Polea dentada T5 36 10mm	13	11,7	50	585
Polea dentada T5 20 16mm	9	8,1	150	1215
Polea dentada T5 20 10mm	9	8,1	200	1620
Polea dentada T5 24 10mm	11	9,9	50	495
Polea dentada T5 24 16mm	11	9,9	50	495
Poleas tensoras	9	8,1	100	810
Correa dentada T5-350 6mm	7,4	6,66	50	333
Correa dentada T5-400 6mm	7,48	6,732	150	1009,8
Correa dentada T5-420 6mm	7,48	6,732	50	336,6
Correa dentada T5-350 8mm	9,44	8,496	50	424,8
ELECTRÓNICA				
Servomotor Tower Pro MG995	14,12	12,71	100	1.270,80
Servomotor Tower Pro MG958	14,95	13,46	150	2.018,25
Servomotor Tower Pro MG959	32	28,8	50	1.440
Placa Arduino Mega 2560	42,98	38,68	50	1.934,10
Fuente alimentación	20	18,00	50	900,00
Teléfono móvil BQ Aquaris E4.5	150	135,00	50	6.750,00
Diodos	0,04	0,04	300	10,80
Módulo Bluetooth HC06	5,49	4,94	50	247,05
TOTAL				36.529,2€

Una vez obtenido el coste del conjunto de materiales necesarios para construcción de las unidades estimadas del brazo articulado si lo dividimos por el número de unidades que se podrán construir obtenemos:

$$\text{Coste del material para un producto} = \frac{\text{coste total material}}{\text{unidades}} = \frac{35.853,75}{50} = 717,07 \frac{\text{€}}{\text{producto}} \quad [17]$$

14.2.2. Coste de montaje

Es el coste de mano de obra dedicada al montaje de partes mecánicas del brazo articulado, ensamblaje de los componentes electrónicos y cableados del brazo robot.

Se considera que un trabajador será capaz de realizar el montaje de un brazo articulado en 10 horas. Teniendo en cuenta que la persona que realice esta tarea cobrará 10€ la hora, se obtiene el siguiente coste:

$$1 \text{ brazo robot articulado} * 10 \frac{\text{horas}}{\text{brazo robot articulado}} * 10 \frac{\text{€}}{\text{hora}} = 100 \frac{\text{€}}{\text{producto}} \quad [18]$$

14.2.3. Coste de amortización

En el coste de amortización se pretende valorar el coste de amortización de los costes de desarrollo y prototipado entre el conjunto de unidades que se pretenden vender.

Se obtiene entonces, el siguiente coste:

$$\begin{aligned} \text{Coste de amortización para un producto} &= \frac{\text{costes de ingeniería} + \text{costes de prototipo}}{\text{unidades}} = \frac{34.798,80}{50} \\ &= 695,97 \frac{\text{€}}{\text{producto}} \quad [19] \end{aligned}$$

14.2.4. Precio final de venta al público

Sumando los tres costes anteriores se obtiene que el precio por unidad de brazo articulado es:

$$717,07 + 100 + 695,97 = 1.513,05 \text{ €}$$

A este precio de coste se le aplica un margen de beneficios del 30%:

$$1.513,05 * 1,30 = 1.966,96€$$

De este modo la empresa recibe un beneficio de 453,91€ por brazo articulado. Finalmente se aplica el I.V.A del 21% para obtener el precio final de venta al público:

$$1815,65 * 1,21 = \mathbf{2.380,02 €}$$

15. Conclusiones

El resultado del trabajo ha sido sumamente satisfactorio debido al éxito en casi todos los aspectos de éste.

El diseño mecánico de las partes da una visión más física sobre el funcionamiento de cualquier mecanismo teniendo en cuenta cada una de sus partes y piezas para ejecutar posteriormente el control.

Por otro lado en la parte principal de este trabajo como ha sido la aplicación móvil, se ha podido ver la ejecución de ésta partiendo del objetivo de controlar un mecanismo desde una interfaz. El lenguaje de programación de la aplicación no ha resultado sencillo debido al poco conocimiento que se tenía de este, pero el programa de diseño unido a la facilidad de adquirir los conocimientos necesarios por los autores se ha realizado la interfaz de una manera eficiente.

Juntamente con la aplicación se programó el microcontrolador para poder entender las instrucciones de la interfaz y ejecutarlas de manera correcta, lo cual ha sido muy interesante poder definir un sistema de comunicación entre ambos dispositivos. En cuanto a los objetivos iniciales, el único que no se ha podido abordar en la finalización del proyecto ha sido el movimiento de las instrucciones teaching a través del eje de coordenadas. La complejidad de las funciones a programar con Arduino ha hecho que no se haya podido llevar a la práctica esta funcionalidad.

También cabe decir que la selección de los componentes adecuados tanto electrónicamente como mecánicamente ha resultado muy satisfactorio en el momento de realizar el montaje del prototipo y una vez ajustado poder observar cómo todos los componentes funcionan correctamente.

A continuación se presenta el aspecto final que muestra el brazo articulado (ilustraciones 99 y 100):

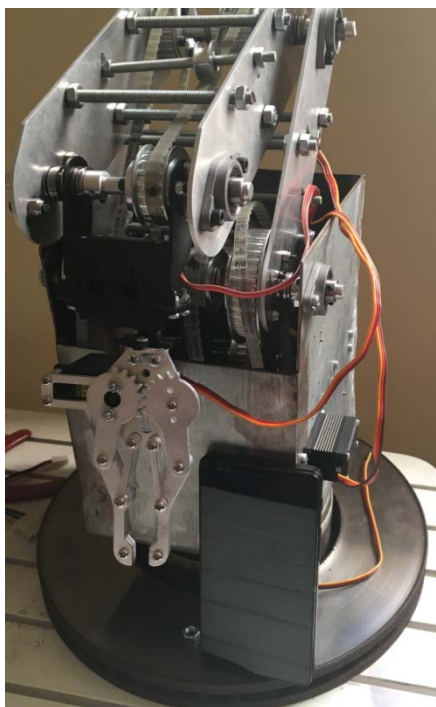


Ilustración 99: Aspecto final del brazo articulado (I)

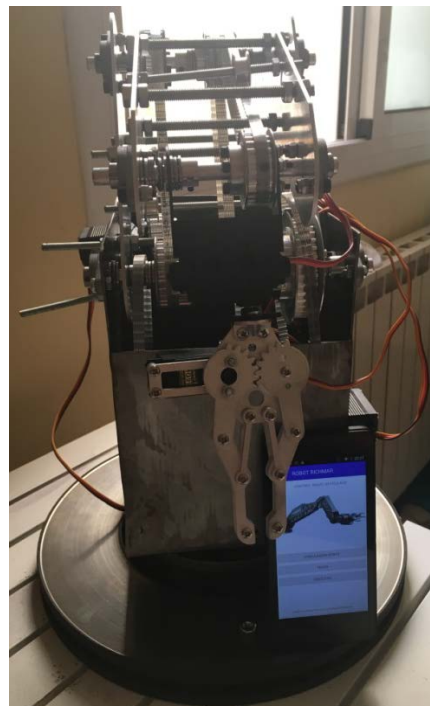


Ilustración 100: Aspecto final del brazo articulado (II)

En cuanto a la distribución de las tareas por parte de los autores, aunque ambos han tenido influencia en todas, ha sido necesario realizar una distribución y separación del trabajo.

Por sus conocimientos de mecánica, Ricard, se ha encargado del diseño mecánico del prototipo. Por otro lado, Marc, ha sido quién ha desarrollado la aplicación con Android Studio. En cuanto a la programación de Arduino ambos autores han tenido la misma influencia, teniendo en cuenta que tanto la aplicación de Android Studio como la programación de la placa Arduino han ido a la par. En la construcción del prototipo, ambos autores han trabajado en equipo consiguiendo un satisfactorio trabajo. Por último, en la elaboración de la memoria y anexos del trabajo cada autor se ha encargado en más proporción de la parte que ha desarrollado.

Por último, en cuanto a mejoras que se podrían aplicar al brazo articulado, aparte de la resolución de la problemática del movimiento del extremo respecto el eje de coordenadas de la base, se consideraría necesario tener un mejor control de las trayectorias. También sería muy interesante que el recorrido entre dos puntos guardados pudiese ser siguiendo un determinado movimiento, por ejemplo, circular.



16. Bibliografía

Bibliografía de consulta

- [I] Android Studio [En línea]. Disponible en: <https://developer.android.com/index.html> [Último acceso: abril 2017]
- [II] Arduino [En línea]. Disponible en: <https://www.arduino.cc/> [Último acceso: abril 2017]
- [III] Arduino Forum [En línea]. Disponible en: <https://forum.arduino.cc/> [Último acceso: abril 2017]
- [IV] Arduino Tinkerkit Braccio [En línea]. Disponible en: <http://www.arduino.org/products/tinkerkit/arduino-tinkerkit-braccio> [Último acceso: abril 2017]
- [V] Microcontroladores [En línea]. Disponible en: <https://microcontroladoresv.wordpress.com/empresas-fabricantes-de-microcontroladores/> [Último acceso: abril 2017]
- [VI] Raspberry [En línea]. Disponible en: <https://www.raspberrystore.com/pt/microcontrolador-xcore-multinucleo-para-raspberry-pi/> [Último acceso: abril 2017]
- [VII] Arduino Mega 2560 R3 [En línea]. Disponible en: <http://arduino.cl/arduino-mega-2560/> [Último acceso: abril 2017]
- [VIII] Bluetooth [En línea]. Disponible en: <https://www.bluetooth.com/what-is-bluetooth-technology> [Último acceso: abril 2017]
- [IX] Fritzing [En línea]. Disponible en: <http://fritzing.org/learning/> [Último acceso: abril 2017]
- [X] Tutoriales Prometec [En línea]. Disponible en: <http://www.prometec.net/timers/> [Último acceso: abril 2017]

[XI] La representación de Denavit Hartenberg. [En línea]. Disponible en: https://personal.us.es/jcortes/Material/Material_archivos/Articulos%20PDF/RepresentDH.pdf [Último acceso: abril 2017].

[XII] Programación de robots [En línea]. Disponible en: <https://automaticaindustrial.wordpress.com/robotica/programacion-de-robot/> [Último acceso: abril 2017]

[XIII] Diseño de transmisión por engranajes [En línea]. Disponible en: <http://www.monografias.com/trabajos82/disenio-transmision-engranajes-rectos/disenio-transmision-engranajes-rectos2.shtml> [Último acceso: abril 2017]

[XIV] Genymotion [En línea]. Disponible en: <https://www.genymotion.com/> [Último acceso: abril 2017]

[XV] RS Components [En línea]. Disponible en: <https://www.google.es/#q=re+amidata> [Último acceso: abril 2017]

Bibliografía de ilustraciones y tablas

[1] Robot welding [En línea]. Disponible en: <http://www.robot-welding.com/robots.htm> [Último acceso: abril 2017]

[2] KUKA robotics, [En línea]. Disponible en: <https://www.kuka.com/es-es/productos-servicios/sistemas-de-robot/robot-industrial/kr-1000-titan> [Último acceso: abril 2017]

[3] Robotics Bible [En línea]. Disponible en: <http://www.roboticsbible.com/robot-drive-systems.html> [Último acceso: abril 2017]

[4] Mitsubishi Electric [En línea]. Disponible en: <https://us.mitsubishielectric.com/fa/en/products/industrial-robots-melfa/horizontal-type-robot/> [Último acceso: abril 2017]

[5] INTEF: Instituto Nacional de Tecnología Educativa y de Formación de profesorado. [En línea]. Disponible en: http://platea.pntic.mec.es/vgonzale/cyr_0708/archivos/_15/Tema_5.4.htm/ [Último acceso: abril 2017]

[6] OWI Robot [En línea]. Disponible en: <http://www.owirobot.com/robotic-arm-edge-1/> [Último acceso: abril 2017]

- [7] Freepic [En línea]. Disponible en: <http://www.freepik.es/fotos-vectores-gratis/brazo-robotico> [Último acceso: abril 2017]
- [8] Lynx Motion [En línea]. Disponible en: <http://www.lynxmotion.com/c-124-al5a.aspx> [Último acceso: abril 2017]
- [9] Super Robótica [En línea]. Disponible en: <http://www.superrobotica.com/S300100.htm> [Último acceso: abril 2017]
- [10] Robot Geek [En línea]. Disponible en: <https://www.robotgeek.com/rg-snapper-core> [Último acceso: abril 2017]
- [11] Arduino – Braccio [En línea]. Disponible en: <http://www.arduino.org/products/tinkerkit/arduino-tinkerkit-braccio> [Último acceso abril 2017]
- [12] Medias Schaffler [En línea]. Disponible en: http://medias.schaeffler.de/medias/de!hp.ec.br.pr/HK..-2RS*HK0810-2RS?lang=es [Último acceso: abril 2017]
- [13] Lily Bearing Manufacturing [En línea]. Disponible en: <https://www.lily-bearing.com/ball-bearings/miniature-bearing/miniature-thrust-bearings/f8-16m/> [Último acceso: abril 2017]
- [14] Bea Transmisión [En línea]. Disponible en: <http://www.beatransmision.com/es/catalogo-industrial/categoria,resto-de-catalogo/categoria,anillos-de-fijacion/> [Último acceso: abril 2017]
- [15] GRABCAD Community [En línea]. Disponible en: <https://grabcad.com/library/tag/kf108> [Último acceso: abril 2017]
- [16] ATMEL [En línea]. Disponible en: <http://www.atmel.com/> [Último acceso: abril 2017]
- [17] Texas Instruments [En línea]. Disponible en: <https://www.ti.com/> [Último acceso: abril 2017]
- [18] Microchip [En línea]. Disponible en: <http://www.microchip.com/> [Último acceso: abril 2017]
- [19] Intel [En línea]. Disponible en: <http://www.intel.es/content/www/es/es/homepage.html> [Último acceso: abril 2017]
- [20] Arduino UNO [En línea]. Disponible en: <https://www.arduino.cc/en/main/arduinoBoardUno> [Último acceso: abril 2017]

[21] Raspberry PI model B. [En línea]. Disponible en: <https://www.raspberrypi.org/products/model-b/> [Último acceso: abril 2017]

[22] Arduino Mega 2560 [En línea]. Disponible en: <https://www.arduino.cc/en/Main/arduinoBoardMega2560> [Último acceso: abril 2017]

[23] ArduinoBot [En línea]. Disponible en: <http://arduinoobot.pbworks.com/w/page/10175781/Motores%20Servo> [Último acceso: abril 2017]

[24] Bluetooth [En línea]. Disponible en: <https://www.bluetooth.com/> [Último acceso: abril 2017]

[25] Android Studio [En línea]. Disponible en: <https://developer.android.com/studio/index.html> [Último acceso: abril 2017]

Anexo A: Planos del prototipo

Elementos normalizados

Geometría y características de las correas según su paso

Tabla 27. Geometrías de las correas normalizadas de métrica T

Passo - PITCH - Teilung - Pas - Paso	Potenza trasmissibile Power rating Leistungsübertragung Puissance transmissible Potencia transmissible max	Giri/1' Rpm U/min Tr/min Rpm max	Velocità lineare Linear speed Umfangsgeschwindigkeit Vitesse linéaire Velocidad lineal max	Puleggia - Pulley - Scheiben - Poulie - Polea De per rinvii o piegamenti rovesci for jockey pulley or deflection drives für Spannrolle oder Gegenbiegung pour renvois ou pour plâges à l'envers para reenvios o doblamiento al revés min	
T 2,5 	~ 0,5 kW	40000	80 m/s	Ø 7 mm	Ø 18 mm
Trasmissioni per piccoli elettrodomestici - Trasmissioni per cineprese Drives for small household appliances - Camera drives Kleine Haushaltsgeräte - Filmkameras Transmissions pour petits électroménagers - Transmissions pour les caméras Transmisiones para pequeños electrodomésticos - Transmisiones para cámaras					
T 5 	~ 5 kW	40000	80 m/s	Ø 15 mm	Ø 30 mm
Trasmissioni per macchine da ufficio - Elettrodomestici - Macchine utensili e macchine da legno - Comandi e regolazioni in genere Office machinery - Household appliances - Machine tools and wood machinery - Control and regulator drives Büromaschinen - Haushaltsgeräte - Werkzeugmaschinen und Holzmaschinen - Steuer und Regelantriebe Transmissions pour machines de bureau - Electroménagers - Machine-outils et machines à bois - Commandes et réglages en général Transmisiones para máquinas de oficina - Electrodomésticos - Máquinas herramientas y para madera - Mandos y ajustes en general					
T 10 	~ 30 kW	15000	60 m/s	Ø 36 mm	Ø 60 mm
Trasmissioni di macchine utensili - Macchine da legno - Pompe - Compressori e ventilatori - Macchine da stampa - Comandi principali ed ausiliari Machine tools and woodworking machinery - Pumps - Compressors and fans - Printing machinery - Primary and auxiliary control drives Werkzeugmaschinen und Holzbearbeitungsmaschinen - Pumpen - Verdichter und Ventilatoren - Druckereimaschinen - Haupt- und Nebenantriebe Transmissions pour machine-outils et machines à bois - Pompes - Compresseurs et ventilateurs - Machines à imprimer - Commandes principales et auxiliaires Transmisiones para máquinas herramientas y para madera - Bombas - Compresores y ventiladores - Máquinas de imprimir - Mandos principales y auxiliares					
T 20 	~ 120 kW	6000	40 m/s	Ø 92 mm	Ø 120 mm
Trasmissioni pesanti - Trasmissioni per macchine da carta - Pompe - Compressori - Trasportatori a rulli Heavy drives - Paper machinery - Pumps - Compressors - Roller conveyor drives Scheranantriebe - Papiermaschinen - Pumpen - Verdichter - Rollenträger Transmissions lourdes - Machines à papier - Pompes - Compresseurs - Bandes transporteuses à rouleaux Transmisiones pesadas - Máquinas para papeleras - Bombas - Compresores - Cintas transportadoras de cilindros					

Geometría de las poleas de paso T5

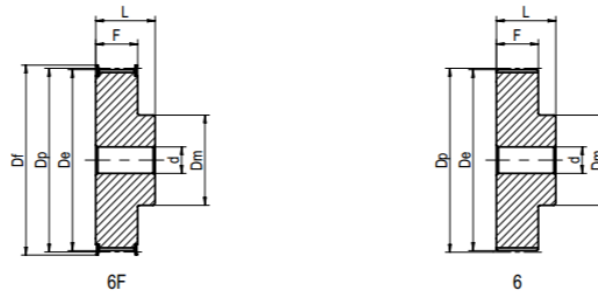


Ilustración 101. Geometría de las poleas dentadas de paso T5

Tabla 28. Valores geométricos según número de dientes en poleas con correas de 10 mm

T 5 - 10 mm		Paso - Pitch - Teilung - Pas - Paso 5 mm Larghezza cinghia - Belt width - Riemenbreite - Largeur de la courroie - Ancho de la correa 10 mm									
Materiale Material Werkstoff Matériel Material	Codice Item number Codierung Code Codigo	Descrizione Designation Bezeichnung Designation Referencia	N° denti No. of teeth Anzahl der Zähne Nombre de dents Cantidad de dientes	Dp	De	Df	Dm	F	L	d	Peso Weight Gewicht Poids Peso
				mm	mm	mm	mm	mm	mm	mm	kg
Aluminio Aluminium Aluminium Aluminium Aluminium	35T051021	21 T 5/10 - 6F	10	15,92	15,05	19,5	8	15	21	-	0,010
	35T051221	21 T 5/12 - 6F	12	19,10	18,25	23,0	11	15	21	-	0,020
	35T051421	21 T 5/14 - 6F	14	22,28	21,45	25,0	14	15	21	-	0,020
	35T051521	21 T 5/15 - 6F	15	23,87	23,05	28,0	16	15	21	6	0,030
	35T051621	21 T 5/16 - 6F	16	25,46	24,60	32,0	18	15	21	6	0,030
	35T051821	21 T 5/18 - 6F	18	28,65	27,80	36,0	20	15	21	6	0,030
	35T051921	21 T 5/19 - 6F	19	30,24	29,40	36,0	22	15	21	6	0,040
	35T052021	21 T 5/20 - 6F	20	31,83	31,00	36,0	23	15	21	6	0,040
	35T052221	21 T 5/22 - 6F	22	35,01	34,25	38,0	24	15	21	6	0,050
	35T052421	21 T 5/24 - 6F	24	38,20	37,40	42,0	26	15	21	6	0,060
	35T052521	21 T 5/25 - 6F	25	39,79	39,00	44,0	26	15	21	6	0,060
	35T052621	21 T 5/26 - 6F	26	41,38	40,60	44,0	26	15	21	6	0,060
	35T052721	21 T 5/27 - 6F	27	42,97	42,20	48,0	30	15	21	8	0,070
	35T052821	21 T 5/28 - 6F	28	44,56	43,75	48,0	32	15	21	8	0,070
	35T053021	21 T 5/30 - 6F	30	47,75	46,95	51,0	34	15	21	8	0,080
	35T053221	21 T 5/32 - 6F	32	50,93	50,10	54,0	38	15	21	8	0,090
	35T053621	21 T 5/36 - 6F	36	57,30	56,45	63,0	38	15	21	8	0,120
	35T054021	21 T 5/40 - 6F	40	63,66	62,85	66,0	40	15	21	8	0,140
	35T054221	21 T 5/42 - 6F	42	66,84	66,00	71,0	40	15	21	8	0,180
	35T054421	21 T 5/44 - 6	44	70,03	69,20	-	45	15	21	8	0,190
35T054821	21 T 5/48 - 6	48	76,39	75,55	-	50	15	21	8	0,200	
35T056021	21 T 5/60 - 6	60	95,49	94,65	-	65	15	21	8	0,310	

Tabla 29. Valores geométricos según número de dientes en poleas con correas de 16 mm

T 5 - 16 mm

Passo - Pitch - Teilung - Pas - Paso 5 mm
Larghezza cinghia - Belt width - Riemenbreite - Largeur de la courroie - Ancho de la correa 16 mm

Materiale Material Werkstoff Matériel Material	Codice Item number Codierung Code Código	Descrizione Designation Bezeichnung Désignation Referencia	N° denti No. of teeth Anzahl der Zähne Nombre de dents Cantidad de dientes	Dp mm	De mm	Df mm	Dm mm	F mm	L mm	d mm	Peso Weight Gewicht Poids Peso kg
Alluminio Aluminum Aluminium Aluminium Aluminio	35T051027	27 T 5/10 - 6F	10	15,92	15,05	19,5	8	21	27	-	0,020
	35T051227	27 T 5/12 - 6F	12	19,10	18,25	23,0	11	21	27	-	0,020
	35T051427	27 T 5/14 - 6F	14	22,28	21,45	25,0	14	21	27	-	0,030
	35T051527	27 T 5/15 - 6F	15	23,87	23,05	28,0	16	21	27	6	0,030
	35T051627	27 T 5/16 - 6F	16	25,46	24,60	32,0	18	21	27	6	0,040
	35T051827	27 T 5/18 - 6F	18	28,65	27,80	32,0	20	21	27	6	0,050
	35T051927	27 T 5/19 - 6F	19	30,24	29,40	36,0	22	21	27	6	0,050
	35T052027	27 T 5/20 - 6F	20	31,83	31,00	36,0	23	21	27	6	0,050
	35T052227	27 T 5/22 - 6F	22	35,01	34,25	38,0	24	21	27	6	0,060
	35T052427	27 T 5/24 - 6F	24	38,20	37,40	42,0	26	21	27	6	0,080
	35T052527	27 T 5/25 - 6F	25	39,79	39,00	44,0	26	21	27	6	0,080
	35T052627	27 T 5/26 - 6F	26	41,38	40,60	44,0	26	21	27	6	0,090
	35T052727	27 T 5/27 - 6F	27	42,97	42,20	48,0	30	21	27	8	0,090
	35T052827	27 T 5/28 - 6F	28	44,56	43,75	48,0	32	21	27	8	0,090
	35T053027	27 T 5/30 - 6F	30	47,75	46,95	51,0	34	21	27	8	0,110
	35T053227	27 T 5/32 - 6F	32	50,93	50,10	54,0	38	21	27	8	0,130
	35T053627	27 T 5/36 - 6F	36	57,30	56,45	63,0	38	21	27	8	0,160
	35T054027	27 T 5/40 - 6F	40	63,66	62,85	66,0	40	21	27	8	0,190
	35T054227	27 T 5/42 - 6F	42	66,84	66,00	71,0	40	21	27	8	0,210
	35T054427	27 T 5/44 - 6	44	70,03	69,20	-	45	21	27	8	0,230
35T054827	27 T 5/48 - 6	48	76,39	75,55	-	50	21	27	8	0,280	
35T056027	27 T 5/60 - 6	60	95,49	94,65	-	65	21	27	8	0,430	

Rodamientos de agujas HK0810

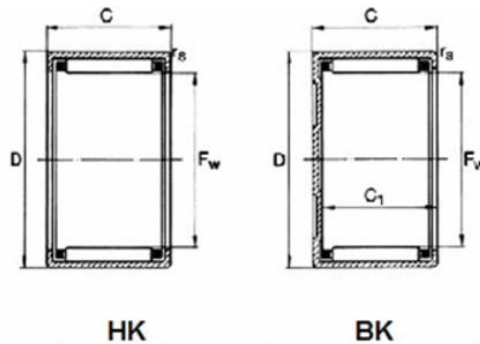


Ilustración 102: Geometría de los rodamientos de agujas HK y BK

Tabla 30. Valores geométricos de los rodamientos de agujas HK y BK.

Diámetro del eje	Designación del transporte			Dimensiones principales			Carga clasificada básica		Velocidad permisible
							Parásitos atmosféricos dinámicos de N		
	HK	BK	Viejo	FW	D	C	C _r	Corazón	RPM
4	HK0408TN			4	8	8	1500	1200	30000
5	HK0508TN		37941/5	5	9	8	1900	1000	27000
	HK0509	BK0509	47941/5	5	9	9	2300	2000	27000
6	HK0608		37941/6	6	10	8	2100	1900	25000
	HK0609	BK0609	47941/6	6	10	9	2500	2400	25000
7	HK0709	BK0709	47941/7	7	11	9	2700	2700	23000
	HK0708		Jul-41	7	12	8			23000
8	HK0808	BK0808	37941/8	8	12	8	2400	2400	20000
	HK0810	BK0810	57941/8	8	12	10	3300	3700	20000
	HK0810		Aug-41	8	14	10			20000
9	HK0908		37941/9	9	13	8	2700	2900	18000
	HK0910	BK0910	57941/9	9	13	10	3700	4400	18000
	HK0912	BK0910	67941/9	9	13	12	4700	5900	18000
10	HK1010	BK1010	57941/10	10	14	10	3900	4800	16000
	HK1012	BK1012	67941/10	10	14	12	4900	6400	16000
	HK1015	BK1015		10	14	15	5800	8000	16000
	HK1010		Oct-41	10	16	10			16000
	HK1012		Oct-42	10	16	12			16000
12	HK1210	BK1210	57941/12	12	16	10	4300	5600	13000
	HK1212	BK1212	67941/12	12	16	12	5300	7500	13000
	HK1214		77941/12	12	16	14	6300	9400	13000
	HK1212		Dec-41	12	17	12			13000
	HK1215		Dec-42	12	17	15			13000
	HK1218		Dec-43	12	17	18			13000
	HKH1212	BKH212	37942/12	12	18	12	6500	7300	13000
	HKH1216		57942/12	12	18	16	8200	10800	16000
13	HK1312	BK1312		13	19	12			1200
14	HK1412	BK1412	37941/14	14	20	12	6300	8100	11000
15	HK1512	BK1512	37941/15	15	21	12	6600	8700	11000
	HK1516	BK1516	57941/14	15	21	16	9400	13700	11000
	HK1522	BK1522		15	21	22	11900	13700	11000
	HK1512		7941/15	15	20	12			11000
	HK1516		7942/15	15	20	16			11000
	HK2020		7943/15	15	20	20			
16	HK1612	BK1612	37941/16	16	22	12	6800	9300	10000
	HK1616	BK1616	57941/16	16	22	16	9700	14600	10000

Soportes con rodamiento de bolas KFL08

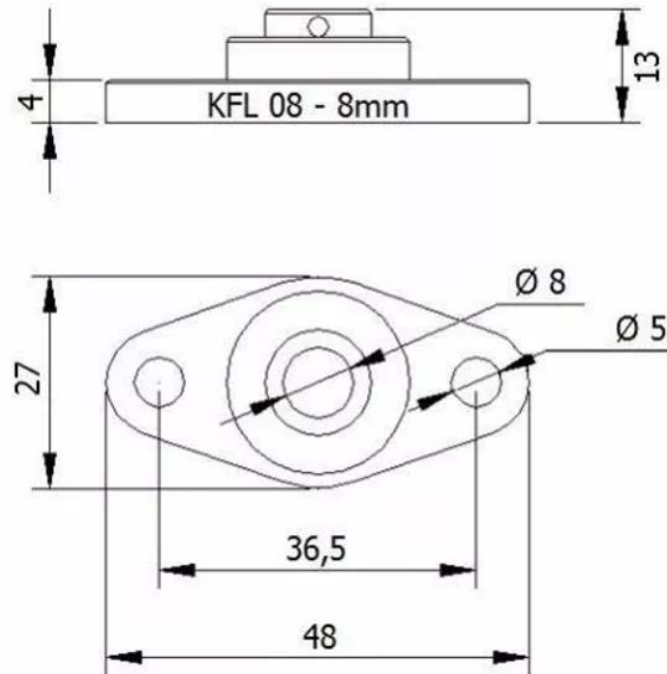


Ilustración 103. Soporte KFL 08 acotado.

Rodamientos de empuje axial F8-16M

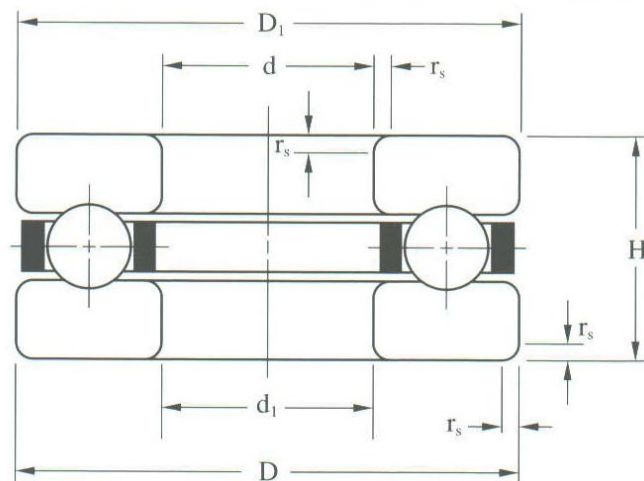


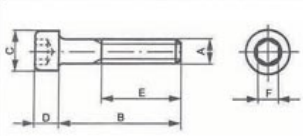
Ilustración 104. Geometría rodamientos de empuje serie Fx

Tabla 31. Valores geométricos de los rodamientos de empuje serie Fx

Type	d	D	d1	D1	H	Load Rating Cr(N)	Load Rating Cor(N)	steel ball	
								piece	diameter
F2-6	2	6	2	6	3	117	83	6	1
F2X-7	2.5	7	2.5	7	3.5	156	117	6	1.2
F3-8	3	8	3.2	7.8	3.5	600	480	6	1.588
F4-9	4	9	4.2	8.8	4	800	520	6	1.588
F4-10	4	10	4.2	9.8	4	658	580	6	1.588
F5-11	5	11	5.2	10.8	4.5	988	880	7	1.588
F6-12	6	12	6.2	11.8	4.5	1600	1255	8	2
F7-15	7	15	7.2	14.8	5	2200	2000	8	2.5
F8-16	8	16	8.2	15.8	5	2500	3000	9	3
F9-17	9	17	9.2	16.8	5	578	627	10	2.381
F10-18	10	18	10.2	17.8	5.5	2230	2721	10	2.381

Tornillerías métricas

Tabla 32. Valores de los tornillos según su métrica

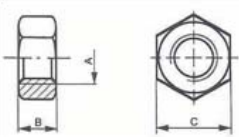


A	M1,6	M2	M2,5	M3	M4	M5	M6	M8	M10	M12	M14	M16	M18	M20	M22	M24	M27	M30	M33	M36	M39	M42	M48
Paso	0,35	0,4	0,45	0,5	0,7	0,8	1	1,25	1,5	1,75	2	2	2,5	2,5	2,5	3	3	3,5	3,5	4	4	4,5	5
E (mín.)	15	16	17	18	20	22	24	28	32	36	40	44	48	52	56	60	66	72	78	84	90	96	108
C (máx.)	3	3,8	4,5	5,5	7	8,5	10	13	16	18	21	24	27	30	33	36	40	45	50	54	58	63	72
D (máx.)	1,6	2	2,5	3	4	5	6	8	10	12	14	16	18	20	22	24	27	30	33	36	39	42	48
F	1,5	1,5	2	2,5	3	4	5	6	8	10	12	14	14	17	17	19	19	22	24	27	27	32	36

(Todas las medidas están expresadas en mm.)

Tuercas métricas

Tabla 33. Valores de las tuercas según su métrica



A	M2	M3	M4	M5	M6	M7	M8	M10	M12	M14	M16	M18	M20	M22	M24	M27	M30	M33	M36	M39	M42	M45	M48	M52	M56	M60	M64	M68
Paso	0,4	0,5	0,7	0,8	1	1	1,25	1,5	1,75	2	2	2,5	2,5	2,5	3	3	3,5	3,5	4	4	4,5	4,5	5	5	5,5	5,5	6	6
B	1,6	2,4	3,2	4	5	5,5	6,5	8	10	11	13	15	16	18	19	22	24	26	29	31	34	36	38	42	45	48	51	54
C	4	5,5	7	8	10	11	13	17	19	22	24	27	30	32	36	41	46	50	55	60	65	70	75	80	85	90	95	100

(Todas las medidas están expresadas en mm.)

Planos

A continuación se presentan los planos de las piezas principales de este proyecto y que han sido diseñadas por los autores.

Para el plano 4 y 5 se deben observar las siguientes tablas para completar la información ya que se usarán para varios elementos.

Por otro lado es importante mencionar que todos los agujeros que aparecen en los planos no han sido referenciados con tolerancias debido a que se comentara aquí que todos estos han sido realizados con tolerancias H7.

Tabla 34. Valores para los ejes estructurales del plano Num. 5. (SOPORTES ESTRUCTURA Y EJES DE POLEAS TENSORAS)

D	L	Cantidad
5 mm	94 mm	1
5 mm	126 mm	2
5 mm	155 mm	3
6 mm	94 mm	3
6 mm	126 mm	3

Tabla 35. Valores para los ejes de las articulaciones, acompaña al plano Núm. 4

L	Cantidad
113 mm	1
145 mm	1
175 mm	1

PLANO 1. BRAZO UNIÓN HOMBRO-CODO



PLANO 2. BRAZO UNIÓN CODO-MUÑECA



PLANO 3. BASE BRAZO ROBÓTICO



PLANO 4. EJES ARTICULACIONES



PLANO 5. EJES ESTRUCTURALES Y DE POLEAS TENSORAS



PLANO 6. POLEA DE 24 DIENTES



PLANO 7. POLEA DE 36 DIENTES



PLANO 8. POLEA DE 20 DIENTES



Anexo B: Código Arduino

A continuación se presenta el código utilizado para la programación de la placa de control Arduino

Mega 2560:

```
//declaración de librerías
#include <String.h>
#include <PWM.h>
#include <Servo.h>

//declaración de variables

int counter_cintura=0; //contador cintura 1 teaching
int counter_cintura1=0;
int counter_hombro=0;
int counter_hombro1=0;
int counter_codo=0;
int counter_codo1=0;
int counter_muneca=0;
int counter_muneca1=0;
int counter_mano=0;
int counter_mano1=0;
int count=0; //contador auxiliar

static int count_posicionar_cintura=0; //contador cintura executor
static int count_posicionar_hombro=0;
static int count_posicionar_codo=0;
static int count_posicionar_muneca=0;
static int count_posicionar_mano=0;

char paso_cintura; //valor del dutty cycle del punto del recorrido al
que se mueve el motor
char paso_hombro;
char paso_codo;
char paso_muneca;
char paso_mano;

char kcintura; //variable auxiliar cintura
char khombro;
char kcodo;
char kmuneca;
char kmano;

int vector_cintura[4]; //vector para inicio de movimiento teaching
progresivo de la cintura
int vector_hombro[4];
int vector_codo[4];
int vector_muneca[4];
int vector_mano[4];

bool mover_cintura; //mover cintura al recibir instrucción teaching
bool mover_hombro;
bool mover_codo;
bool mover_muneca;
bool mover_mano;
```

```
int flag;          //flags de condiciones
int flag_recorrido=0;

char cabeza[2];   //cabecera paquetes de datos
int primer_caracter; //primer dato del paquete
int segundo_caracter; //segundo dato del paquete
int tercer_caracter; //tercer dato del paquete
char cabeza_final[2]; //final paquete de datos

bool nueva_instruccion=false; //nueva instrucción recibida
bool instruccion_teach=false; //instrucción teach recibida
bool instruccion_executor=false; //instrucción executor recibida
bool instruccion_pinza=false; //instrucción posiciones pinza recibida
bool instruccion_longitud=false; //instrucción longitud movimiento
teaching recibida
bool inicio_recorrido=false; //iniciar recorrido
bool inicio_movimiento=false; //inicio movimiento JOINT/XYZ
bool inicio_posicionamiento=false; //inicio posicionamiento
servomotores para realizar recorrido
bool continuacion=false; //iniciar movimiento servomotor pinza

bool posicionar_cintura=false; //posicionar cintura en recorrido executor
bool posicionar_hombro=false;
bool posicionar_codo=false;
bool posicionar_muneca=false;
bool posicionar_mano=false;
bool posicionar_pinza=false;
bool primer_punto=false;

int pos_cintura_punto; //duty cycle que debe recibir el motor de la
cintura en un punto del recorrido
int pos_hombro_punto;
int pos_codo_punto;
int pos_muneca_punto;
int pos_mano_punto;

int i=0; //variables auxiliares
int c=0;
int d=0;

int longitud_movimiento=0; //inicialización longitud de movimiento
int valor=5; //valor de la longitud de movimiento

int DC_motor_cintura=250; //Posicion inicial motor cintura
int DC_motor_hombro=250;
int DC_motor_codo=250;
int DC_motor_muneca=250;
int DC_motor_mano=250;
int DC_motor_pinza=60;

int DC_cintura_P1; //Posiciones guardadas de la cintura en los
diferentes puntos
int DC_cintura_P2;
int DC_cintura_P3;
int DC_cintura_P4;

int DC_hombro_P1; //Posiciones guardadas del hombro en los diferentes
puntos
int DC_hombro_P2;
```

```

int DC_hombro_P3;
int DC_hombro_P4;

int DC_codo_P1;      //Posiciones guardadas del codo en los diferentes
puntos
int DC_codo_P2;
int DC_codo_P3;
int DC_codo_P4;

int DC_muneca_P1;   //Posiciones guardadas de la muñeca en los
diferentes puntos
int DC_muneca_P2;
int DC_muneca_P3;
int DC_muneca_P4;

int DC_mano_P1;     //Posiciones guardadas de la mano en los diferentes
puntos
int DC_mano_P2;
int DC_mano_P3;
int DC_mano_P4;

int punto1;        //Punto 1 guardado con la aplicacion
int punto2;
int punto3;
int punto4;

int pinza1=1;      //inicialización de los estados de la pinza en cada
punto
int pinza2=1;
int pinza3=1;
int pinza4=1;

const int motor_cintura=7;    //Pin digital de conexión del motor
cintura
const int motor_hombro=8;
const int motor_codo=11;
const int motor_muneca=12;
const int motor_mano=6;
const int motor_pinza=5;

int vector_acel[4]={1, 4, 9, 16}; //Vector de aceleraciones para el
recorrido

//inicio void set up - instrucciones que solo se ejecutan al inicio de
la aplicación
void setup()
{
    InitTimers();                //Inicialización de los timers
    //Serial1.begin(9600);        //Velocidad del puerto del módulo
Bluetooth
    Serial.begin(9600);          //Abrimos la comunicación serie con el
PC y establecemos velocidad
    Serial.flush();

    bool success_timer1=SetPinFrequencySafe(11,50); //pin 11/12 timer1
    bool success_timer2=SetPinFrequency(9,50);      //pin 9/10 timer2
    bool success_timer3=SetPinFrequencySafe(2,50);  //pin 2/3/5 timer3

```

```

    bool success_timer4=SetPinFrequencySafe(6,50);    //pin 6/7/8 timer4

    analogWrite(motor_cintura,DC_motor_cintura);    //movemos motor cintura
a su posición inicial
    analogWrite(motor_hombro,DC_motor_hombro);    //movemos motor hombro
a su posición inicial
    analogWrite(motor_codo,DC_motor_codo);    //movemos motor codo a
su posición inicial
    analogWrite(motor_muneca,DC_motor_muneca);    //movemos motor muñeca
a su posición inicial
    analogWrite(motor_mano,DC_motor_mano);    //movemos motor muñeca
a su posición inicial
    analogWrite(motor_pinza,DC_motor_pinza);    //movemos motor pinza a
su posición inicial
}

// inicio void loop - instrucciones que se ejecutan repetidamente en el
programa
void loop()
{
    Serial.flush();
    char dato_recibido=0;

    if (inicio_recorrido==true)
    {
        iniciar_recorrido();
    }

    movimiento_cintura();    //Invocación de las funciones del
movimiento de cada servo repetidamente para que siempre mantengan la
posición
    movimiento_hombro();
    movimiento_codo();
    movimiento_muneca();
    movimiento_mano();
    movimiento_pinza();

    if(Serial.available(<1)return;
    dato_recibido=Serial.read();    //lectura por el puerto serie para
recibir los paquetes de datos

//FILTRO INSTRUCCIONES TEACH
if (instruccion_teach==true)    //cuando ya se ha recibido una instrucción
teach se compara si es una instrucción JOINT, X,Y,X o de guardar
posiciones
{
    if((c>4)&&(c<7)) {    //comprobación del final del paquete de
datos
        if (dato_recibido=='B'){
            cabecera_final[i]=dato_recibido;
            Serial.print(cabecera_final[i]);
            i++;
            c++;
        }
        else
            c=0;
    }

    if (c==7){
        if ((cabecera_final[0]=='B')&&(cabecera_final[1]=='B')){

```

```

        i=0;
        c=0;
        nueva_instruccion=true; //activación nueva instrucción
        if (nueva_instruccion==true){
            Serial.println("INSTRUCCION TEACH ENVIADA");
            movimiento_servos(); //al tratarse de una instrucción
teach se llama a la función servos() que permitirá el movimiento de los
motores
            nueva_instruccion=false; //desactivamos nueva
instrucción
            instruccion_teach=false; //activamos instrucción
teach
        }
    }
    else{
        c=0;
        i=0;
    }
}

if (c==4){ //almacenamiento tercer caracter del paquete de
datos
    tercer_caracter=dato_recibido-48; //
    c++;
    if (tercer_caracter>2){
        c=0;
        flag=0;
    }
    Serial.print(tercer_caracter);
}
if (c==3){ //almacenamiento segundo caracter del paquete de
datos
    segundo_caracter=dato_recibido-48;
    c++;
    switch (flag)
    {
        case 1: //si el segundo dato es un
            if (segundo_caracter>3){
                c=0;
                flag=0;
            }
            break;
        case 2:
        case 3:
            if (segundo_caracter>4){
                c=0;
                flag=0;
            }
            break;
        default:
            segundo_caracter=0;
            c=0;
            break;
    }
    Serial.print(segundo_caracter);
}
}
}
//FILTRO INSTRUCCIONES LONGITUD

```

```

if (instruccion_longitud==true){ //cuando ya se ha recibido una
instrucción de tipo de longitud de movimiento se modifica el valor de la
variable de longitud de movimiento de las instruccion de teach que se
reciban
if((c>3)&&(c<6)) {
    if (dato_recibido=='B'){ //comprobación del final del paquete
de datos
        cabecera_final[i]=dato_recibido;
        Serial.print(cabecera_final[i]);
        i++;
        c++;
    }
    else
    c=0;
}

if (c==6){
    if ((cabecera_final[0]=='B')&&(cabecera_final[1]=='B')){
        i=0;
        c=0;
        nueva_instruccion=true; //nueva instruccion recibida
        if (nueva_instruccion==true){
            Serial.println("INSTRUCCION LONGITUD ENVIADA");
            ajuste_longitud_movimiento();
            nueva_instruccion=false;
            instruccion_longitud=false;
        }
    }
    else{
        c=0;
        i=0;
    }
}

if (c==3){ //almacenamiento del tipo de longitud de movimiento
corto - medio - largo
    longitud_movimiento=dato_recibido-48;
    c++;
    Serial.print(longitud_movimiento);
}

//FILTRO INSTRUCCIONES PINZA //cuando ya se ha recibido una instrucción
pinza se guarda el estado de la pinza en cada uno de los puntos a los
que puede acceder el brazo articulado
if (instruccion_pinza==true)
{
    if((c>6)&&(c<9)) { //comprobación del final del paquete de
datos
        if (dato_recibido=='B'){
            cabecera_final[i]=dato_recibido;
            Serial.print(cabecera_final[i]);
            i++;
            c++;
        }
        else
        c=0;
    }
}

```

```

    if (c==9){
        if ((cabecera_final[0]=='B')&&(cabecera_final[1]=='B')){
            i=0;
            c=0;
            nueva_instruccion=true; //nueva instrucción recibida
            if (nueva_instruccion==true){
                Serial.println("INSTRUCCION PINZA ENVIADA");
                nueva_instruccion=false;
                instruccion_pinza=false;
            }
        }
        else{
            c=0;
            i=0;
        }
    }

    if (c==6){
        pinza4=dato_recibido-48;
        c++;
        Serial.print(pinza4); //estado de la pinza en el punto 4 del
espacio guardado mediante teaching
    }

    if (c==5){
        pinza3=dato_recibido-48;
        c++;
        Serial.print(pinza3); //estado de la pinza en el punto 3 del
espacio guardado mediante teaching
    }

    if (c==4){
        pinza2=dato_recibido-48;
        c++;
        Serial.print(pinza2); //estado de la pinza en el punto 2 del
espacio guardado mediante teaching
    }

    if (c==3){
        pinza1=dato_recibido-48;
        c++;
        Serial.print(pinza1); //estado de la pinza en el punto 1 del
espacio guardado mediante teaching
    }
}
//FILTRO INSTRUCCIONES EXECUTOR

if (instruccion_executor==true) //cuando ya se ha recibido una
instrucción executor se guarda el orden al que se desea realizar el
recorrido y se comprueba el final del paquete de datos
{
    if((c>6)&&(c<9)) { //comprobación del final del paquete de datos
        if (dato_recibido=='B'){
            cabecera_final[i]=dato_recibido;
            Serial.print(cabecera_final[i]);
            i++;
            c++;
        }
    }
}

```



```

    }
    else
    c=0;
}

if (c==9){
    if ((cabecera_final[0]=='B')&&(cabecera_final[1]=='B')){
        i=0;
        c=0;
        nueva_instruccion=true; //nueva instrucción recibida
        if (nueva_instruccion==true){
            Serial.println("INSTRUCCION EXECUTOR ENVIADA");
            nueva_instruccion=false;
            instruccion_executor=false;
            inicio_recorrido=true; //realizamos inicio del
recorrido
                inicio_posicionamiento=true;
            }
        }
        else{
            c=0;
            i=0;
        }
    }
}

if (c==6){
    punto4=dato_recibido-48;
    c++;
    Serial.print(punto4); //se guarda el cuarto punto al que el
brazo articulado debe finalizar el recorrido
}

if (c==5){
    punto3=dato_recibido-48;
    c++;
    Serial.print(punto3); //se guarda el tercer punto al que el
brazo articulado debe acceder durante el recorrido
}

if (c==4){
    punto2=dato_recibido-48;
    c++;
    Serial.print(punto2); //se guarda el segundo punto al que el
brazo articulado debe acceder durante el recorrido
}

if (c==3){
    punto1=dato_recibido-48; //se guarda el primer punto al que
el brazo articulado debe iniciar el recorrido
    c++;
    Serial.print(punto1);
}
}

if (c==2){ //comprobación del primer valor del paquete de
datos que se ha recibido
    primer_caracter=dato_recibido-48;
}

```

```

switch (primer_caracter)
{
    case 1:      //si el primer dato es un 1 se trata de una
instrucción teach
        flag=1;
        c++;
        instruccion_teach=true;
        instruccion_executor=false;
        instruccion_pinza=false;
        instruccion_longitud=false;
        Serial.print(primer_caracter);
        break;
    case 2:      //si el primer dato es un 2 se trata de una
instrucción teach
        flag=2;
        c++;
        instruccion_teach=true;
        instruccion_executor=false;
        instruccion_pinza=false;
        instruccion_longitud=false;
        Serial.print(primer_caracter);
        break;
    case 3:      //si el primer dato es un 3 se trata de una
instrucción teach
        flag=3;
        c++;
        instruccion_teach=true;
        instruccion_executor=false;
        instruccion_pinza=false;
        instruccion_longitud=false;
        Serial.print(primer_caracter);
        break;
    case 4:
instrucción executor
        flag=4; // si el primer dato es un 4 se trata de una
        c++;
        instruccion_executor=true;
        instruccion_teach=false;
        instruccion_pinza=false;
        instruccion_longitud=false;
        Serial.print(primer_caracter);
        break;
    case 5:      // si el primer dato es un 5 se trata de una
instruccion de posicionamiento de la pinza
        flag=5;
        c++;
        instruccion_pinza=true;
        instruccion_executor=false;
        instruccion_teach=false;
        instruccion_longitud=false;
        Serial.print(primer_caracter);
        break;
    case 6:      //si el primer dato es un 6 se trata de una
instrucción de longitud de movimiento
        flag=6;
        c++;
        instruccion_longitud=true;
        instruccion_executor=false;
        instruccion_teach=false;
        instruccion_pinza=false;

```

```

        Serial.print(primer_caracter);

        break;
        default: //en caso contrario no hacemos nada
            flag=0;
            c=0;
            Serial.print(primer_caracter);
        break;
        Serial.print(flag);
    }
}
if (c<2) { //comprobación del inicio de la cabeza
    if (dato_recibido=='A'){
        cabeza[c]=dato_recibido;
        Serial.write(cabeza[c]);
        c++;
    }
    else{
        c=0;
    }
}

/*if(Serial.available())
    Serial.write(Serial.read());*/
}

/////////////////////////////////////////FUNCIONES PARA EL MOVIMIENTO DE CADA
MOTOR/////////////////////////////////////////

void movimiento_cintura() //FUNCION MOVIMIENTO CINTURA
{
    if (mover_cintura){ //inicialización de un primer contador para que
        la variación del dutty cycle de la señal del servo no aumente de manera
        directa
        counter_cintura++;
        if (counter_cintura==500){ //cuando el primer contador llega a 500 se
        resetea y se activa un segundo timer con contara hasta 3
        counter_cintura=0;
        counter_cintural++;
        }
        if (counter_cintural==2){ //cuando el segundo timer se ve
        incrementado el dutty cycle de la señal del servomotor se incremente de
        manera proporcional al vector_cintura
        counter_cintural=0;
        kcintura++;
        Serial.println(vector_cintura[kcintura]);
        }
        DC_motor_cintura=vector_cintura[kcintura]; //una vez el segundo
        contador llega a 3 el incremento del dutty cycle de la señal ya puede
        ser constante y se ha conseguido una arrancada progresiva y no directa
        if (kcintura==3){
            kcintura=0;
            mover_cintura=0;
        }
    }
}
    analogWrite(motor_cintura,DC_motor_cintura); //aplicamos la señal PWM
al servomotor de manera constante
}

```

```

void movimiento_hombro() //FUNCION MOVIMIENTO HOMBRO
{
    if (mover_hombro){ //inicialización de un primer contador para que la
    variación del dutty cycle de la señal del servo no aumente de manera
    directa
        counter_hombro++;

        if (counter_hombro==500){ //cuando el primer contador llega a 500 se
        resetea y se activa un segundo timer con contara hasta 3
            counter_hombro=0;
            counter_hombro1++;
        }
        if (counter_hombro1==2){ //cuando el segundo timer se ve
        incrementado el dutty cycle de la señal del servomotor se incremente de
        manera proporcional al vector_hombro
            counter_hombro1=0;
            khombro++;
            Serial.println(vector_hombro[khombro]);
        }
        // Serial.println(vector_muneca[k]);
        DC_motor_hombro=vector_hombro[khombro]; //una vez el segundo contador
        llega a 3 el incremento del dutty cycle de la señal ya puede ser
        constante y se ha conseguido una arrancada progresiva y no directa
        if (khombro==3){
            khombro=0;
            mover_hombro=0;
        }
        }
        analogWrite(motor_hombro,DC_motor_hombro); //aplicamos la señal PWM
        al servomotor de manera constante
    }

void movimiento_codo() //FUNCION MOVIMIENTO CODO
{
    if (mover_codo){ //inicialización de un primer contador para que la
    variación del dutty cycle de la señal del servo no aumente de manera
    directa
        counter_codo++;

        if (counter_codo==500){ //cuando el primer contador llega a 500 se
        resetea y se activa un segundo timer con contara hasta 3
            counter_codo=0;
            counter_codo1++;
        }
        if (counter_codo1==2){ //cuando el segundo timer se ve incrementado
        el dutty cycle de la señal del servomotor se incremente de manera
        proporcional al vector_codo
            counter_codo1=0;
            kcodo++;
            Serial.println(vector_muneca[kcodo]);
        }
        // Serial.println(vector_muneca[k]);
        DC_motor_codo=vector_codo[kcodo]; //una vez el segundo contador llega
        a 3 el incremento del dutty cycle de la señal ya puede ser constante y
        se ha conseguido una arrancada progresiva y no directa
        if (kcodo==3){
            kcodo=0;
            mover_codo=0;
        }
        }
}

```

```

    }
    analogWrite(motor_codo,DC_motor_codo); //aplicamos la señal PWM al
servomotor de manera constante
}

void movimiento_muneca() //FUNCION MOVIMIENTO MUNECA
{
    if (mover_muneca){ //inicialización de un primer contador para que la
variación del dutty cycle de la señal del servo no aumente de manera
directa
        counter_muneca++;

        if (counter_muneca==500){ //cuando el primer contador llega a 500 se
resetea y se activa un segundo timer con contara hasta 3
            counter_muneca=0;
            counter_munecal++;
        }
        if (counter_munecal==2){ //cuando el segundo timer se ve
incrementado el dutty cycle de la señal del servomotor se incremente de
manera proporcional al vector_muñeca
            counter_munecal=0;
            kmuneca++;
            Serial.println(vector_muneca[kmuneca]);
        }
        // Serial.println(vector_muneca[k]);
        DC_motor_muneca=vector_muneca[kmuneca]; //una vez el segundo contador
llega a 3 el incremento del dutty cycle de la señal ya puede ser
constante y se ha conseguido una arrancada progresiva y no directa
        if (kmuneca==3){
            kmuneca=0;
            mover_muneca=0;
        }
    }
    analogWrite(motor_muneca,DC_motor_muneca); //aplicamos la señal PWM
al servomotor de manera constante
}

void movimiento_mano() //FUNCION MOVIMIENTO MANO
{
    if (mover_mano){ //inicialización de un primer contador para que la
variación del dutty cycle de la señal del servo no aumente de manera
directa
        counter_mano++;

        if (counter_mano==500){ //cuando el primer contador llega a 500 se
resetea y se activa un segundo timer con contara hasta 3
            counter_mano=0;
            counter_manol++;
        }
        if (counter_manol==2){ //cuando el segundo timer se ve incrementado
el dutty cycle de la señal del servomotor se incremente de manera
proporcional al vector_mano
            counter_manol=0;
            kmano++;
            Serial.println(vector_muneca[kmano]);
        }
        // Serial.println(vector_muneca[k]);
        DC_motor_muneca=vector_muneca[kmano];
        if (kmano==3){
            kmano=0;

```

```

        mover_mano=0;
    }
}
    analogWrite(motor_mano,DC_motor_mano); //aplicamos la señal PWM al
servomotor de manera constante
}

void movimiento_pinza() //FUNCION MOVIMIENTO PINZA
{
    analogWrite(motor_pinza,DC_motor_pinza); //aplicamos la señal PWM al
servomotor de manera constante
}

/////////////////////////////////////////FUNCION PARA AJUSTAR LONGITUD DE
MOVIMIENTO/////////////////////////////////////////
void ajuste_longitud_movimiento()
{
    switch (longitud_movimiento)
    {
        case 0:
            valor=5;
            Serial.println(valor);
            break;
        case 1:
            valor=10;
            Serial.println(valor);
            break;
        case 2:
            valor=20;
            Serial.println(valor);
            break;
        default:
            //no hacemos nada
            break;
    }
}

/////////////////////////////////////////FUNCION PARA MOVER SERVOS EN X,Y,Z, JOINT Y GUARDAR
POSICIONES/////////////////////////////////////////
void movimiento_servos()
{
    switch (primer_caracter) //filtro del primer caracter guardado
anteriormente para diferenciar entre instrucción JOINT, XYZ o guardar
posiciones
    {
        case 1: //si el primer caracter es un 1 se trata de movimiento por
sistema de coordonedas XYZ
            switch (segundo_caracter)
            {
                case 1:

                    break;
                case 2:

                    break;
                case 3:

```

```

        break;
        default:

            break;
    }
    break;

    case 2: //si el primer caracter es un 2 se trata de un movimiento
por sistema JOINT
        switch (segundo_caracter)
        {
            case 1://en caso de que el segundo caracter sea un 1 será un
movimiento de la cintura
                if (tercer_caracter==0){ //si tercer caracter es un 0 sera
realizara un decremento de la posición del servomotor de la cintura
                    vector_cintura[0]= DC_motor_cintura+5; //uso de un vector
para hacer un arranque del movimiento de la cintura progresivo
                    vector_cintura[1]= DC_motor_cintura+10;
                    vector_cintura[2]= DC_motor_cintura+15;
                    vector_cintura[3]= DC_motor_cintura+20;
                    mover_cintura=1;
                }else{ //si tercer caracter es un 1 sera
realizara un incremento de la posición del servomotor de la cintura
                    vector_cintura[0]= DC_motor_cintura-5; //uso de un vector
para hacer un arranque del movimiento de la cintura progresivo
                    vector_cintura[1]= DC_motor_cintura-10;
                    vector_cintura[2]= DC_motor_cintura-15;
                    vector_cintura[3]= DC_motor_cintura-20;
                    mover_cintura=1;
                }
            break;
            case 2://en caso de que el segundo caracter sea un 1 será un
movimiento del hombro
                if (tercer_caracter==0){ //si tercer caracter es un 0 sera
realizara un decremento de la posición del servomotor del hombro
                    vector_hombro[0]=DC_motor_hombro+5; //uso de un vector para
hacer un arranque del movimiento del hombro progresivo
                    vector_hombro[1]=DC_motor_hombro+10;
                    vector_hombro[2]=DC_motor_hombro+15;
                    vector_hombro[3]=DC_motor_hombro+20;
                    mover_hombro=1;
                }else{ //si tercer caracter es un 1 sera
realizara un incremento de la posición del servomotor del hombro
                    vector_hombro[0]=DC_motor_hombro-5; //uso de un vector para
hacer un arranque del movimiento del hombro progresivo
                    vector_hombro[1]=DC_motor_hombro-10;
                    vector_hombro[2]=DC_motor_hombro-15;
                    vector_hombro[3]=DC_motor_hombro-20;
                    mover_hombro=1;
                }
            break;
            case 3://en caso de que el segundo caracter sea un 1 será un
movimiento del codo
                if (tercer_caracter==0){ //si tercer caracter es un 0 sera
realizara un decremento de la posición del servomotor del codo
                    vector_codo[0]=DC_motor_codo+5; //uso de un vector para
hacer un arranque del movimiento del codo progresivo
                    vector_codo[1]=DC_motor_codo+10;
                    vector_codo[2]=DC_motor_codo+15;
                    vector_codo[3]=DC_motor_codo+20;

```

```

        mover_codo=1;
    }else{ //si tercer caracter es un 1 sera
realizara un incremento de la posición del servomotor del codo
        vector_codo[0]=DC_motor_codo-5; //uso de un vector para
hacer un arranque del movimiento del codo progresivo
        vector_codo[1]=DC_motor_codo-10;
        vector_codo[2]=DC_motor_codo-15;
        vector_codo[3]=DC_motor_codo-20;
        mover_codo=1;
    }
    break;
    case 4://en caso de que el segundo caracter sea un 1 será un
movimiento de la muñeca
        if (tercer_caracter==0){ //si tercer caracter es un 0 sera
realizara un decremento de la posición del servomotor de la muñeca
            vector_muneca[0]=DC_motor_muneca+5; //uso de un vector para
hacer un arranque del movimiento de la muñeca progresivo
            vector_muneca[1]=DC_motor_muneca+10;
            vector_muneca[2]=DC_motor_muneca+15;
            vector_muneca[3]=DC_motor_muneca+20;
            mover_muneca=1;
        }else{ //si tercer caracter es un 1 sera
realizara un incremento de la posición del servomotor de la muñeca
            vector_muneca[0]=DC_motor_muneca-5; //uso de un vector para
hacer un arranque del movimiento de la muñeca progresivo
            vector_muneca[1]=DC_motor_muneca-10;
            vector_muneca[2]=DC_motor_muneca-15;
            vector_muneca[3]=DC_motor_muneca-20;
            mover_muneca=1;
        }
        case 5://en caso de que el segundo caracter sea un 1 será un
movimiento de la mano
            if (tercer_caracter==0){ //si tercer caracter es un 0 sera
realizara un decremento de la posición del servomotor de la mano
                vector_mano[0]=DC_motor_mano+5; //uso de un vector para
hacer un arranque del movimiento de la mano progresivo
                vector_mano[1]=DC_motor_mano+10;
                vector_mano[2]=DC_motor_mano+15;
                vector_mano[3]=DC_motor_mano+20;
                mover_mano=1;
            }else{ //si tercer caracter es un 1 sera
realizara un incremento de la posición del servomotor de la mano

                vector_mano[0]=DC_motor_mano-5; //uso de un vector para
hacer un arranque del movimiento de la mano progresivo
                vector_mano[1]=DC_motor_mano-10;
                vector_mano[2]=DC_motor_mano-15;
                vector_mano[3]=DC_motor_mano-20;
                mover_mano=1;
            }

        break;
    default:
// en caso contrario no se realiza nada
    break;
}

break;

```



```

    case 3: //si el segundo caracter es un 3 se guardaran las posiciones
en las que se encuentran los servomotores
        switch (segundo_caracter)
        {
            case 1: //guardamos la posicion del punto 1 de todos los
servomotores
                DC_cintura_P1 = DC_motor_cintura;
                DC_hombro_P1 = DC_motor_hombro;
                DC_codo_P1 = DC_motor_codo;
                DC_muneca_P1 = DC_motor_muneca;
                DC_mano_P1 = DC_motor_mano;
                Serial.println("POSICION 1 GUARDADA");
                Serial.println(DC_hombro_P1);

                break;
            case 2: //guardamos la posicion del punto 2 de todos los
servomotores
                DC_cintura_P2=DC_motor_cintura;
                DC_hombro_P2 = DC_motor_hombro;
                DC_codo_P2 = DC_motor_codo;
                DC_muneca_P2 = DC_motor_muneca;
                DC_mano_P2 = DC_motor_mano;
                Serial.println("POSICION 2 GUARDADA");

                break;
            case 3: //guardamos la posicion del punto 3 de todos los
servomotores
                DC_cintura_P3=DC_motor_cintura;
                DC_hombro_P3 = DC_motor_hombro;
                DC_codo_P3 = DC_motor_codo;
                DC_muneca_P3 = DC_motor_muneca;
                DC_mano_P3 = DC_motor_mano;
                Serial.println("POSICION 3 GUARDADA");

                break;
            case 4: //guardamos la posicion del punto 4 de todos los
servomotores
                DC_cintura_P4=DC_motor_cintura;
                DC_hombro_P4 = DC_motor_hombro;
                DC_codo_P4 = DC_motor_codo;
                DC_muneca_P4 = DC_motor_muneca;
                DC_mano_P4 = DC_motor_mano;
                Serial.println("POSICION 4 GUARDADA");

                break;
            default:
                // no hacemos nada
                break;
        }
        break;
    }
}

/////////////////////////////////////////FUNCION PARA REALIZAR
RECORRIDO/////////////////////////////////////////
void iniciar_recorrido (void) //inicio del recorrido que ha introducido
el usuario
{
    if (flag_recorrido==0){
        switch (puntol)

```

```

{
  case 0: //no haremos nada ya que no hay recorrido
  inicio_recorrido=false;
  flag_recorrido=0;
  break;

  case 1: //la primera posición a la que quiere
acceder el usuario es el punto 1 guardado con anterioridad
  inicio_movimiento=true;
  Serial.println("MOVIENDO A PUNTO 1");
  pos_cintura_punto=DC_cintura_P1; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
  pos_hombro_punto=DC_hombro_P1;
  pos_codo_punto=DC_codo_P1;
  pos_muneca_punto=DC_muneca_P1;
  pos_mano_punto=DC_mano_P1;
  if (inicio_posicionamiento==true)
  {
    posicionar_cintura(); //posicionamiento de cada unos de los
servomotores
    posicionar_hombro();
    posicionar_codo();
    posicionar_muneca();
    posicionar_mano();
    Serial.println("entrado");
  }

  if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodigo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
  {
    inicio_posicionamiento=false;
    posicionarmano=false;
    posicionarmuneca=false;
    posicionarhombro=false;
    posicionarcodigo=false;
    posicionarcintura=false;
    continuacion=true;
  }

  if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
    posicionar_pinza(pinza1);
  }
  if (posicionarpinza==true)
  {
    flag_recorrido=1;
    Serial.println("PUNTO 1 ALCANZADO");
    posicionarpinza=false;
    inicio_posicionamiento=true;
    //inicio_movimiento=false;
    //final_movimiento=false;
  }

  break;

  case 2: //la primera posición a la que quiere acceder el
usuario es el punto 2 guardado con anterioridad
  inicio_movimiento=true;

```

```

    Serial.println("MOVIENDO A PUNTO 1");
    pos_cintura_punto=DC_cintura_P2;    //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
    pos_hombro_punto=DC_hombro_P2;
    pos_codo_punto=DC_codo_P2;
    pos_muneca_punto=DC_muneca_P2;
    pos_mano_punto=DC_mano_P2;
    if (inicio_posicionamiento==true)
    {
        posicionar_cintura();    //posicionamiento de cada unos de los
servomotores
        posicionar_hombro();
        posicionar_codo();
        posicionar_muneca();
        posicionar_mano();
        Serial.println("entrado");
    }

    if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
    {
        inicio_posicionamiento=false;
        posicionarmano=false;
        posicionarmuneca=false;
        posicionarhombro=false;
        posicionarcodo=false;
        posicionarcintura=false;
        continuacion=true;
    }

    if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
        posicionar_pinza(pinza2);
    }
    if (posicionarpinza==true)
    {
        flag_recorrido=1;
        Serial.println("PUNTO 1 ALCANZADO");
        posicionarpinza=false;
        inicio_posicionamiento=true;
        //inicio_movimiento=false;
        //final_movimiento=false;
    }
    break;

    case 3: //la primera posición a la que quiere acceder el
usuario es el punto 3 guardado con anterioridad
        inicio_movimiento=true;
        Serial.println("MOVIENDO A PUNTO 1");
        pos_cintura_punto=DC_cintura_P3;    //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
        pos_hombro_punto=DC_hombro_P3;
        pos_codo_punto=DC_codo_P3;
        pos_muneca_punto=DC_muneca_P3;
        pos_mano_punto=DC_mano_P3;
        if (inicio_posicionamiento==true)
        {
            posicionar_cintura();    //posicionamiento de cada unos de los
servomotores

```

```

    posicionar_hombro();
    posicionar_codo();
    posicionar_muneca();
    posicionar_mano();
    Serial.println("entrado");
}

if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
{
    inicio_posicionamiento=false;
    posicionarmano=false;
    posicionarmuneca=false;
    posicionarhombro=false;
    posicionarcodo=false;
    posicionarcintura=false;
    continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
    posicionar_pinza(pinza3);
}
if (posicionarpinza==true)
{
    flag_recorrido=1;
    Serial.println("PUNTO 1 ALCANZADO");
    posicionarpinza=false;
    inicio_posicionamiento=true;
    //inicio_movimiento=false;
    //final_movimiento=false;
}
break;

case 4: //la primera posición a la que quiere acceder
el usuario es el punto 4 guardado con anterioridad
    inicio_movimiento=true;
    Serial.println("MOVIENDO A PUNTO 4");
    pos_cintura_punto=DC_cintura_P4; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
    pos_hombro_punto=DC_hombro_P4;
    pos_codo_punto=DC_codo_P4;
    pos_muneca_punto=DC_muneca_P4;
    pos_mano_punto=DC_mano_P4;
    if (inicio_posicionamiento==true)
    {
        posicionar_cintura(); //posicionamiento de cada unos de los
servomotores
        posicionar_hombro();
        posicionar_codo();
        posicionar_muneca();
        posicionar_mano();
        Serial.println("entrado");
    }

    if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
{

```

```

    inicio_posicionamiento=false;
    posicionarmano=false;
    posicionarmuneca=false;
    posicionarhombro=false;
    posicionarcodigo=false;
    posicionarcintura=false;
    continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
    posicionar_pinza(pinza4);
}
if (posicionarpinza==true)
{
flag_recorrido=1;
Serial.println("PUNTO 1 ALCANZADO");
posicionarpinza=false;
inicio_posicionamiento=true;
//inicio_movimiento=false;
//final_movimiento=false;
}
break;

default:
//no hacemos nada
break;
}

}

if (flag_recorrido==1){
switch (punto2)
{
case 0: //no haremos nada ya que no hay recorrido
inicio_recorrido=false;
flag_recorrido=0;
break;

case 1: //la segunda posición a la que quiere acceder el usuario
durante el recorrido es el punto 1 guardado con anterioridad
inicio_movimiento=true;
Serial.println("MOVIENDO A PUNTO 2");
pos_cintura_punto=DC_cintura_P1; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
pos_hombro_punto=DC_hombro_P1;
pos_codo_punto=DC_codo_P1;
pos_muneca_punto=DC_muneca_P1;
pos_mano_punto=DC_mano_P1;
if (inicio_posicionamiento==true)
{
posicionar_cintura(); //posicionamiento de cada unos de los
servomotores
posicionar_hombro();
posicionar_codo();
posicionar_muneca();
posicionar_mano();
Serial.println("entrado");
}
}
}

```

```

    if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
    {
        inicio_posicionamiento=false;
        posicionarmano=false;
        posicionarmuneca=false;
        posicionarhombro=false;
        posicionarcodo=false;
        posicionarcintura=false;
        continuacion=true;
    }

    if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
        posicionar_pinza(pinza1);
    }
    if (posicionarpinza==true)
    {
        flag_recorrido=2;
        Serial.println("PUNTO 2 ALCANZADO");
        posicionarpinza=false;
        inicio_posicionamiento=true;
        //inicio_movimiento=false;
        //final_movimiento=false;
    }
    break;

    case 2: //la segunda posición a la que quiere acceder el
usuario durante el recorrido es el punto 2 guardado con anterioridad
        inicio_movimiento=true;
        Serial.println("MOVIENDO A PUNTO 2");
        pos_cintura_punto=DC_cintura_P2; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
        pos_hombro_punto=DC_hombro_P2;
        pos_codo_punto=DC_codo_P2;
        pos_muneca_punto=DC_muneca_P2;
        pos_mano_punto=DC_mano_P2;
        if (inicio_posicionamiento==true)
        {
            posicionar_cintura(); //posicionamiento de cada unos de los
servomotores
            posicionar_hombro();
            posicionar_codo();
            posicionar_muneca();
            posicionar_mano();
            Serial.println("entrado");
        }

        if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
    {
        inicio_posicionamiento=false;
        posicionarmano=false;
        posicionarmuneca=false;
        posicionarhombro=false;
        posicionarcodo=false;
        posicionarcintura=false;
    }

```

```

    continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
    posicionar_pinza(pinza2);
}
if (posicionarpinza==true)
{
flag_recorrido=2;
Serial.println("PUNTO 2 ALCANZADO");
posicionarpinza=false;
inicio_posicionamiento=true;
//inicio_movimiento=false;
//final_movimiento=false;
}
break;

case 3: //la segunda posición a la que quiere acceder el
usuario durante el recorrido es el punto 3 guardado con anterioridad
inicio_movimiento=true;
Serial.println("MOVIENDO A PUNTO 2");
pos_cintura_punto=DC_cintura_P3; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
pos_hombro_punto=DC_hombro_P3;
pos_codo_punto=DC_codo_P3;
pos_muneca_punto=DC_muneca_P3;
pos_mano_punto=DC_mano_P3;
if (inicio_posicionamiento==true)
{
posicionar_cintura(); //posicionamiento de cada unos de los
servomotores
posicionar_hombro();
posicionar_codo();
posicionar_muneca();
posicionar_mano();
Serial.println("entrado");
}

if((posicionar muneca==true)&&(posicionar hombro==true)&&(posicionar cin
tura==true)&&(posicionar codo==true)&&(posicionar mano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
{
inicio_posicionamiento=false;
posicionar_mano=false;
posicionar_muneca=false;
posicionar_hombro=false;
posicionar_codo=false;
posicionar_cintura=false;
continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
    posicionar_pinza(pinza3);
}
if (posicionarpinza==true)
{
flag_recorrido=2;
Serial.println("PUNTO 2 ALCANZADO");
}

```

```

posicionarpinza=false;
inicio_posicionamiento=true;
//inicio_movimiento=false;
//final_movimiento=false;
}
break;

case 4: //la segunda posición a la que quiere acceder el
usuario durante el recorrido es el punto 4 guardado con anterioridad
inicio_movimiento=true;
Serial.println("MOVIENDO A PUNTO 2");
pos_cintura_punto=DC_cintura_P4; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
pos_hombro_punto=DC_hombro_P4;
pos_codo_punto=DC_codo_P4;
pos_muneca_punto=DC_muneca_P4;
pos_mano_punto=DC_mano_P4;
if (inicio_posicionamiento==true)
{
posicionar_cintura(); //posicionamiento de cada unos de los
servomotores
posicionar_hombro();
posicionar_codo();
posicionar_muneca();
posicionar_mano();
Serial.println("entrado");
}

if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodigo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
{
inicio_posicionamiento=false;
posicionarmano=false;
posicionarmuneca=false;
posicionarhombro=false;
posicionarcodigo=false;
posicionarcintura=false;
continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
posicionar_pinza(pinza4);
}
if (posicionarpinza==true)
{
flag_recorrido=2;
Serial.println("PUNTO 2 ALCANZADO");
posicionarpinza=false;
inicio_posicionamiento=true;
//inicio_movimiento=false;
//final_movimiento=false;
}
break;

default:
//no hacemos nada
break;
}

```



```

}
if (flag_recorrido==2){
switch (punto3)
{
case 0:          //no haremos nada ya que no hay recorrido
inicio_recorrido=false;
flag_recorrido=0;
break;

case 1:          //la tercera posición a la que quiere acceder el
usuario durante el recorrido es el punto 1 guardado con anterioridad
inicio_movimiento=true;
Serial.println("MOVIENDO A PUNTO 3");
pos_cintura_punto=DC_cintura_P1;    //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
pos_hombro_punto=DC_hombro_P1;
pos_codo_punto=DC_codo_P1;
pos_muneca_punto=DC_muneca_P1;
pos_mano_punto=DC_mano_P1;
if (inicio_posicionamiento==true)
{
posicionar_cintura();    //posicionamiento de cada unos de los
servomotores
posicionar_hombro();
posicionar_codo();
posicionar_muneca();
posicionar_mano();
Serial.println("entrado");
}

if((posicionar_muneca==true)&&(posicionar_hombro==true)&&(posicionar_cin
tura==true)&&(posicionar_codo==true)&&(posicionar_mano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
{
inicio_posicionamiento=false;
posicionar_mano=false;
posicionar_muneca=false;
posicionar_hombro=false;
posicionar_codo=false;
posicionar_cintura=false;
continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
posicionar_pinza(pinza1);
}
if (posicionar_pinza==true)
{
flag_recorrido=3;
Serial.println("PUNTO 3 ALCANZADO");
posicionar_pinza=false;
inicio_posicionamiento=true;
//inicio_movimiento=false;
//final_movimiento=false;
}
break;

case 2:          //la tercera posición a la que quiere acceder el
usuario durante el recorrido es el punto 2 guardado con anterioridad

```

```

    inicio_movimiento=true;
    Serial.println("MOVIENDO A PUNTO 3");
    pos_cintura_punto=DC_cintura_P2;    //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
    pos_hombro_punto=DC_hombro_P2;
    pos_codo_punto=DC_codo_P2;
    pos_muneca_punto=DC_muneca_P2;
    pos_mano_punto=DC_mano_P2;
    if (inicio_posicionamiento==true)
    {
        posicionar_cintura();    //posicionamiento de cada unos de los
servomotores
        posicionar_hombro();
        posicionar_codo();
        posicionar_muneca();
        posicionar_mano();
        Serial.println("entrado");
    }

    if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodo==true)&&(posicionarmano==true))    //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
    {
        inicio_posicionamiento=false;
        posicionarmano=false;
        posicionarmuneca=false;
        posicionarhombro=false;
        posicionarcodo=false;
        posicionarcintura=false;
        continuacion=true;
    }

    if (continuacion==true){    //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
        posicionar_pinza(pinza2);
    }
    if (posicionarpinza==true)
    {
        flag_recorrido=3;
        Serial.println("PUNTO 3 ALCANZADO");
        posicionarpinza=false;
        inicio_posicionamiento=true;
        //inicio_movimiento=false;
        //final_movimiento=false;
    }
    break;

    case 3:    //la tercera posición a la que quiere acceder el
usuario durante el recorrido es el punto 3 guardado con anterioridad
        inicio_movimiento=true;
        Serial.println("MOVIENDO A PUNTO 3");
        pos_cintura_punto=DC_cintura_P3;    //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
        pos_hombro_punto=DC_hombro_P3;
        pos_codo_punto=DC_codo_P3;
        pos_muneca_punto=DC_muneca_P3;
        pos_mano_punto=DC_mano_P3;
        if (inicio_posicionamiento==true)
        {

```

```

    posicionar_cintura();      //posicionamiento de cada unos de los
servomotores
    posicionar_hombro();
    posicionar_codo();
    posicionar_muneca();
    posicionar_mano();
    Serial.println("entrado");
}

if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
{
    inicio_posicionamiento=false;
    posicionarmano=false;
    posicionarmuneca=false;
    posicionarhombro=false;
    posicionarcodo=false;
    posicionarcintura=false;
    continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
    posicionar_pinza(pinza3);
}
if (posicionarpinza==true)
{
    flag_recorrido=3;
    Serial.println("PUNTO 3 ALCANZADO");
    posicionarpinza=false;
    inicio_posicionamiento=true;
    //inicio_movimiento=false;
    //final_movimiento=false;
}
break;

case 4:          //la tercera posición a la que quiere acceder el
usuario durante el recorrido es el punto 4 guardado con anterioridad
    inicio_movimiento=true;
    Serial.println("MOVIENDO A PUNTO 3");
    pos_cintura_punto=DC_cintura_P4;      //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
    pos_hombro_punto=DC_hombro_P4;
    pos_codo_punto=DC_codo_P4;
    pos_muneca_punto=DC_muneca_P4;
    pos_mano_punto=DC_mano_P4;
    if (inicio_posicionamiento==true)
    {
        posicionar_cintura();      //posicionamiento de cada unos de los
servomotores
        posicionar_hombro();
        posicionar_codo();
        posicionar_muneca();
        posicionar_mano();
        Serial.println("entrado");
    }

if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin

```

```

tura==true)&&(posicionarcodo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
{
    inicio_posicionamiento=false;
    posicionarmano=false;
    posicionarmuneca=false;
    posicionarhombro=false;
    posicionarcodo=false;
    posicionarcintura=false;
    continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
    posicionar_pinza(pinza4);
}
if (posicionarpinza==true)
{
    flag_recorrido=3;
    Serial.println("PUNTO 3 ALCANZADO");
    posicionarpinza=false;
    inicio_posicionamiento=true;
    //inicio_movimiento=false;
    //final_movimiento=false;
}
break;

default:
//no hacemos nada
break;
}
}
if (flag_recorrido==3){
switch (punto4)
{
    case 0: //no haremos nada ya que no hay recorrido
        inicio_recorrido=false;
        flag_recorrido=0;
        break;

    case 1: //la última posición a la que quiere acceder el
usuario durante el recorrido es el punto 1 guardado con anterioridad
        inicio_movimiento=true;
        Serial.println("MOVIENDO A PUNTO 4");
        pos_cintura_punto=DC_cintura_P1; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
        pos_hombro_punto=DC_hombro_P1;
        pos_codo_punto=DC_codo_P1;
        pos_muneca_punto=DC_muneca_P1;
        pos_mano_punto=DC_mano_P1;
        if (inicio_posicionamiento==true)
        {
            posicionar_cintura(); //posicionamiento de cada unos de los
servomotores
            posicionar_hombro();
            posicionar_codo();
            posicionar_muneca();
            posicionar_mano();
            Serial.println("entrado");
        }
}
}

```

```

    if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
    {
        inicio_posicionamiento=false;
        posicionarmano=false;
        posicionarmuneca=false;
        posicionarhombro=false;
        posicionarcodo=false;
        posicionarcintura=false;
        continuacion=true;
    }

    if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
        posicionar_pinza(pinza1);
    }
    if (posicionarpinza==true)
    {
        flag_recorrido=0;
        inicio_recorrido=false;
        Serial.println("PUNTO 4 ALCANZADO");
        posicionarpinza=false;
        inicio_posicionamiento=true;
        //inicio_movimiento=false;
        //final_movimiento=false;
    }
    break;

    case 2: //la última posición a la que quiere acceder el usuario
durante el recorrido es el punto 2 guardado con anterioridad
        inicio_movimiento=true;
        Serial.println("MOVIENDO A PUNTO 4");
        pos_cintura_punto=DC_cintura_P2; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
        pos_hombro_punto=DC_hombro_P2;
        pos_codo_punto=DC_codo_P2;
        pos_muneca_punto=DC_muneca_P2;
        pos_mano_punto=DC_mano_P2;
        if (inicio_posicionamiento==true)
        {
            posicionar_cintura(); //posicionamiento de cada unos de los
servomotores
            posicionar_hombro();
            posicionar_codo();
            posicionar_muneca();
            posicionar_mano();
            Serial.println("entrado");
        }

        if((posicionarmuneca==true)&&(posicionarhombro==true)&&(posicionarcin
tura==true)&&(posicionarcodo==true)&&(posicionarmano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
    {
        inicio_posicionamiento=false;
        posicionarmano=false;
        posicionarmuneca=false;
        posicionarhombro=false;
        posicionarcodo=false;
    }

```

```

    posicionar_cintura=false;
    continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
    posicionar_pinza(pinza2);
}
if (posicionar_pinza==true)
{
    flag_recorrido=0;
    inicio_recorrido=false;
    Serial.println("PUNTO 4 ALCANZADO");
    posicionar_pinza=false;
    inicio_posicionamiento=true;
    //inicio_movimiento=false;
    //final_movimiento=false;
}
break;

case 3: //la última posición a la que quiere acceder el usuario
durante el recorrido es el punto 3 guardado con anterioridad
    inicio_movimiento=true;
    Serial.println("MOVIENDO A PUNTO 4");
    pos_cintura_punto=DC_cintura_P3; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
    pos_hombro_punto=DC_hombro_P3;
    pos_codo_punto=DC_codo_P3;
    pos_muneca_punto=DC_muneca_P3;
    pos_mano_punto=DC_mano_P3;
    if (inicio_posicionamiento==true)
    {
        posicionar_cintura(); //posicionamiento de cada uno de los
servomotores
        posicionar_hombro();
        posicionar_codo();
        posicionar_muneca();
        posicionar_mano();
        Serial.println("entrado");
    }

    if((posicionar_muneca==true)&&(posicionar_hombro==true)&&(posicionar_cin
tura==true)&&(posicionar_codo==true)&&(posicionar_mano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
    {
        inicio_posicionamiento=false;
        posicionar_mano=false;
        posicionar_muneca=false;
        posicionar_hombro=false;
        posicionar_codo=false;
        posicionar_cintura=false;
        continuacion=true;
    }

    if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
        posicionar_pinza(pinza3);
    }
    if (posicionar_pinza==true)
    {

```

```

flag_recorrido=0;
inicio_recorrido=false;
Serial.println("PUNTO 4 ALCANZADO");
posicionarpinza=false;
inicio_posicionamiento=true;
//inicio_movimiento=false;
//final_movimiento=false;
}
break;

case 4: //la última posición a la que quiere acceder el
usuario durante el recorrido es el punto 4 guardado con anterioridad
inicio_movimiento=true;
Serial.println("MOVIENDO A PUNTO 4");
pos_cintura_punto=DC_cintura_P4; //asignación de las posiciones a
las que se deben desplazar todos los servomotores excepto la pinza
pos_hombro_punto=DC_hombro_P4;
pos_codo_punto=DC_codo_P4;
pos_muneca_punto=DC_muneca_P4;
pos_mano_punto=DC_mano_P4;
if (inicio_posicionamiento==true)
{
posicionar_cintura(); //posicionamiento de cada unos de los
servomotores
posicionar_hombro();
posicionar_codo();
posicionar_muneca();
posicionar_mano();
Serial.println("entrado");
}

if((posicionar muneca==true)&&(posicionar hombro==true)&&(posicionar cin
tura==true)&&(posicionar codo==true)&&(posicionar mano==true)) //cuando
todos los servomotores ya se han posicionado se pasa a mover la pinza
{
inicio_posicionamiento=false;
posicionar_mano=false;
posicionar_muneca=false;
posicionar_hombro=false;
posicionar_codo=false;
posicionar_cintura=false;
continuacion=true;
}

if (continuacion==true){ //movimiento de la pinza en función del
estado solicitado con las instrucciones pinza
posicionar_pinza(pinza4);
}
if (posicionarpinza==true)
{
flag_recorrido=0;
inicio_recorrido=false;
Serial.println("PUNTO 4 ALCANZADO");
posicionarpinza=false;
inicio_posicionamiento=true;
//inicio_movimiento=false;
//final_movimiento=false;
}
break;

```

```

    default:
    //no hacemos nada
    break;
}
}
}

void posicionar_cintura() //movimiento de la cinta utilizando un
vector de aceleración y desaceleración para el inicio y el final del
movimiento respectivamente
{
    int temp;

    count_posicionar_cintura++;

    if (pos_cintura_punto>DC_motor_cintura){
        temp=pos_cintura_punto-DC_motor_cintura;

        if (temp>40){
            if ((count_posicionar_cintura==1)&&(paso_cintura<4)){
                DC_motor_cintura=DC_motor_cintura+vector_acel[paso_cintura];
                paso_cintura++;
                count_posicionar_cintura=0;
            }
            if (count_posicionar_cintura==2){
                DC_motor_cintura=DC_motor_cintura+20;
                count_posicionar_cintura=0;
            }
        }
        else{
            if (count_posicionar_cintura==2){
                count_posicionar_cintura=0;
                if(temp<=5){
                    DC_motor_cintura=pos_cintura_punto;
                    paso_cintura=0;
                    posicionar_cintura=true;
                }else{
                    DC_motor_cintura=DC_motor_cintura+5;
                }
            }
        }
    }
    else{
        if (pos_cintura_punto<DC_motor_cintura){
            temp=DC_motor_cintura-pos_cintura_punto;

            if (temp>40){
                if ((count_posicionar_cintura==1)&&(paso_cintura<4)){
                    DC_motor_cintura=DC_motor_cintura+vector_acel[3-
paso_cintura];
                    paso_cintura++;
                    count_posicionar_cintura=0;
                }
                if (count_posicionar_cintura==2){
                    DC_motor_cintura=DC_motor_cintura-20;
                    count_posicionar_cintura=0;
                }
            }
            else{
                if (count_posicionar_cintura==2){

```



```

        count_posicionar_cintura=0;
        if(temp<=5){
            DC_motor_cintura=pos_cintura_punto;
            paso_cintura=0;
            posicionarcintura=true;
        }else{
            DC_motor_cintura=DC_motor_cintura-5;
        }
    }
}
}
}

void posicionar_hombro() //movimiento del hombro utilizando un vector
de aceleración y desaceleración para el inicio y el final del movimiento
respectivamente
{
    int temp;

    count_posicionar_hombro++;

    if (pos_hombro_punto>DC_motor_hombro){
        temp=pos_hombro_punto-DC_motor_hombro;

        if (temp>40){
            if ((count_posicionar_hombro==1)&&(paso_hombro<4)){
                DC_motor_hombro=DC_motor_hombro+vector_acel[paso_hombro];
                paso_hombro++;
                count_posicionar_hombro=0;
            }
            if (count_posicionar_hombro==2){
                DC_motor_hombro=DC_motor_hombro+20;
                count_posicionar_hombro=0;
            }
        }else{
            if (count_posicionar_hombro==2){
                count_posicionar_hombro=0;
                if(temp<=5){
                    DC_motor_hombro=pos_hombro_punto;
                    paso_hombro=0;
                    posicionarhombro=true;
                }else{
                    DC_motor_hombro=DC_motor_hombro+5;
                }
            }
        }
    }else{
        if (pos_hombro_punto<DC_motor_hombro){
            temp=DC_motor_hombro-pos_hombro_punto;

            if (temp>40){
                if ((count_posicionar_hombro==1)&&(paso_hombro<4)){
                    DC_motor_hombro=DC_motor_hombro+vector_acel[3-paso_hombro];
                    paso_hombro++;
                    count_posicionar_hombro=0;
                }
                if (count_posicionar_hombro==2){
                    DC_motor_hombro=DC_motor_hombro-20;
                }
            }
        }
    }
}

```

```

        count_posicionar_hombro=0;
    }

    }else{
        if (count_posicionar_hombro==2){
            count_posicionar_hombro=0;
            if(temp<=5){
                DC_motor_hombro=pos_hombro_punto;
                paso_hombro=0;
                posicionarhombro=true;
            }else{
                DC_motor_hombro=DC_motor_hombro-5;
            }
        }
    }
}
}
}
}

void posicionar_codo() //movimiento del codo utilizando un vector de
aceleración y desaceleración para el inicio y el final del movimiento
respectivamente
{
    int temp;

    count_posicionar_codo++;

    if (pos_codo_punto>DC_motor_codo){
        temp=pos_codo_punto-DC_motor_codo;

        if (temp>40){
            if ((count_posicionar_codo==1)&&(paso_codo<4)){
                DC_motor_codo=DC_motor_codo+vector_acel[paso_codo];
                paso_codo++;
                count_posicionar_codo=0;
            }
            if (count_posicionar_codo==2){
                DC_motor_codo=DC_motor_codo+20;
                count_posicionar_codo=0;
            }
        }
        }else{
            if (count_posicionar_codo==2){
                count_posicionar_codo=0;
                if(temp<=5){
                    DC_motor_codo=pos_codo_punto;
                    paso_codo=0;
                    posicionar_codo=true;
                }else{
                    DC_motor_codo=DC_motor_codo+5;
                }
            }
        }
    }else{
        if (pos_codo_punto<DC_motor_codo){
            temp=DC_motor_codo-pos_codo_punto;

            if (temp>40){
                if ((count_posicionar_codo==1)&&(paso_codo<4)){
                    DC_motor_codo=DC_motor_codo+vector_acel[3-paso_codo];

```



```
    if (DC_motor_pinza<70){
        DC_motor_pinza=DC_motor_pinza+valor;
    }
    if (DC_motor_pinza==70){
        DC_motor_pinza=70;
        posicionarpinza=true;
        continuacion=false;
    }
    break;
case 1:
    if (DC_motor_pinza<400){
        DC_motor_pinza=DC_motor_pinza+valor;
    }
    if (DC_motor_pinza>400){
        DC_motor_pinza=DC_motor_pinza-valor;
    }
    if (DC_motor_pinza==400){
        DC_motor_pinza=400;
        posicionarpinza=true;
        continuacion=false;
    }
    break;
default:
    //no hacemos nada
    break;
}
}
```



Anexo C: Esquema electrónico

Esquema electrónico:

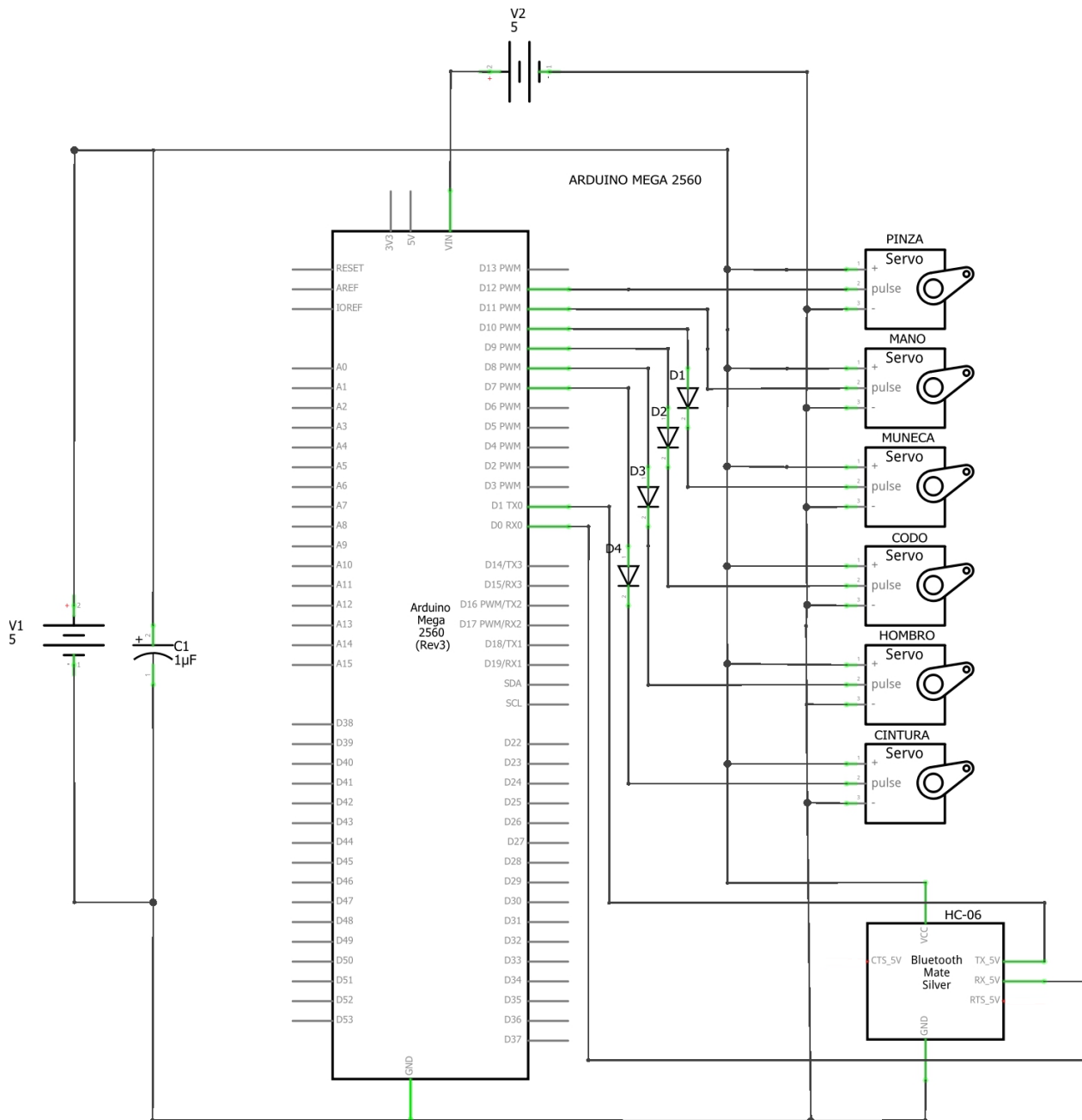


Ilustración 105. Esquema de conexión electrónico

Anexo D: Código Android Studio

A continuación se presenta el código que permite la generación de cada uno de los Layout de cada pantalla juntamente con el código de programación de cada una de las funcionalidades de la aplicación:

Menú principal:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">

<include layout="@layout/toolbar_layout"/>

<TextView
    android:text="Instrucciones de uso y funcionamiento"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView3"
    android:textAlignment="center"
    tools:textColor="@color/colorPrimary"
    android:layout_alignParentBottom="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="15dp"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

<Button
    android:text="VINCULACIÓN ROBOT"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/conexionbluetooth"
    android:layout_above="@+id/teach"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

<Button
    android:text="EXECUTOR"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/executor"
    android:layout_marginBottom="94dp"
    android:layout_above="@+id/textView3"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
```



```

<Button
    android:text="TEACH"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/teach"
    android:layout_above="@+id/executor"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

<TextView
    android:text="CONTROL BRAZO ARTICULADO"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/titulo"
    tools:textColor="@color/colorPrimary"
    tools:textAlignment="center"
    android:layout_marginTop="69dp"
    android:layout_marginLeft="79dp"
    android:layout_marginStart="79dp"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/robot"
    android:id="@+id/imageView"
    android:layout_alignTop="@+id/titulo"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />
</RelativeLayout>

package com.example.maracruzvinjeta.robotrichmar;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.Window;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    Button teach; //variable botón TEACH
    Button executor; //variable botón EXECUTOR
    Button conexionbluetooth; //variable botón vinculación dispositivo
    public static boolean bluetoothemparejado = false; //variable que
    permitira saber si el dispositivo está emparejado

    private Toolbar toolbar; //llamamiento a la toolbar que permitirá
    movernos entre pantallas

    @Override

```

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    toolbar = (Toolbar)findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    executor = (Button) findViewById(R.id.executor); //definir
función del botón EXECUTOR
    executor.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (bluetoothemparejado==false){ //comprobar el
estado de la vinculación con el robot
                Toast.makeText(getApplicationContext(),"NECESARIO
EMPAREJAR ROBOT", Toast.LENGTH_LONG).show(); //advertencia de que el
robot aún no ha sido vinculado
            }
            else {
                Intent controlcoordenadas = new
Intent(MainActivity.this, Main7Activity.class); //acceso a la pantalla
executor aceptado
                startActivity(controlcoordenadas);
            }
        }
    });
    teach = (Button) findViewById(R.id.teach); //definir
función del botón TEACH
    teach.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if (bluetoothemparejado==false){ //comprobar
el estado de la vinculación con el robot
                Toast.makeText(getApplicationContext(),"NECESARIO
EMPAREJAR ROBOT", Toast.LENGTH_LONG).show(); //advertencia de que el
robot aún no ha sido vinculado
            }
            else {
                Intent teach = new Intent(MainActivity.this,
Main6Activity.class); //acceso a pantalla teach aceptado
                startActivity(teach);
            }
        }
    });
    conexionbluetooth = (Button)
findViewById(R.id.conexionbluetooth); //definir función del botón para
emparejar robot
    conexionbluetooth.setOnClickListener(new View.OnClickListener()
{
        @Override
        public void onClick(View v) {
            Intent conexionbluetooth = new Intent(MainActivity.this,
Main4Activity.class); //acceso a la pantalla de vinculación bluetooth
            startActivity(conexionbluetooth);
        }
    });
}

```

```

    }
    @Override
    public boolean onSupportNavigateUp() { //función que permite acceder
a la pantalla anterior en la toolbar
        onBackPressed();
        return false;
    }
}

```

Vinculación robot:

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">

    <include layout="@layout/toolbar_layout"/>

    <Button
android:id="@+id/search"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/Find"
    android:layout_marginBottom="20dp"
    android:layout_above="@+id/listView1"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

    <Button
android:id="@+id/paired"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/List"
    android:layout_above="@+id/search"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

    <Button
android:id="@+id/turnOff"
android:layout_width="208dp"
android:layout_height="wrap_content"
android:text="@string/off"
    android:layout_above="@+id/paired"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true"
    android:layout_toRightOf="@+id/turnOn"
    android:layout_toEndOf="@+id/turnOn" />

    <Button
android:id="@+id/turnOn"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/on"

```

```

        android:layout_above="@+id/paired"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

<ListView
    android:id="@+id/listView1"
    android:layout_width="fill_parent"
    android:layout_height="200dp"
        android:layout_weight="24.40"
        android:layout_marginBottom="59dp"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true">
</ListView>

<TextView
    android:id="@+id/text"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:text="@string/Text"
        android:layout_above="@+id/turnOff"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginLeft="12dp"
        android:layout_marginStart="12dp"
        android:layout_marginBottom="18dp" />

</RelativeLayout>

package com.example.maracruzvineta.robotrichmar;

import android.bluetooth.BluetoothSocket;
import android.os.Bundle;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Set;
import java.util.UUID;
import java.util.logging.Handler;
import android.content.Intent;
import android.content.IntentFilter;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

```

```

public class Main4Activity extends AppCompatActivity {
    private static final int REQUEST_ENABLE_BT = 1;
    private Button onBtn; //boton para activar bluetooth
    private Button offBtn; //boton para desactivar bluetooth
    private Button listBtn; //boton para mostrar lista de dispositivos
emparejados
    private Button findBtn; //boton para mostrar lista de dispositivos
nuevos
    private TextView text;
    private BluetoothAdapter myBluetoothAdapter; //declaraciones
bluetooth
    private Set<BluetoothDevice> pairedDevices; //acceso a
dispositivos ya emparejados
    private ListView myListView; //lista donde de
mostrarán los dispositivos
    private ArrayAdapter<String> BTArrayAdapter;
    public static String EXTRA_ADDRESS = "device_address";

    public String address = null;
    BluetoothSocket btSocket = null;
    private boolean isBtConnected = false;
    static final UUID myUUID = UUID.fromString("00001101-0000-1000-8000-
00805F9B34FB");
    public static ConnectedThread mConnectedThread;

    private Toolbar toolbar;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main4);

        toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        // comprobaciones para saber si el dispositivo móvil dispone de
conexión bluetooth
        myBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
        if (myBluetoothAdapter == null) {
            onBtn.setEnabled(false);
            offBtn.setEnabled(false);
            listBtn.setEnabled(false);
            findBtn.setEnabled(false);
            text.setText("ESTADO: NO SOPORTADO");

            Toast.makeText(getApplicationContext(), "Su teléfono no
dispone de conexión Bluetooth",
                Toast.LENGTH_LONG).show();
        } else {
            text = (TextView) findViewById(R.id.text);
            onBtn = (Button) findViewById(R.id.turnOn);
            onBtn.setOnClickListener(new OnClickListener() {

                @Override
                public void onClick(View v) {

```

```

        // TODO Auto-generated method stub
        on(v);
    }
});
//desactivación a través de la aplicación del bluetooth del
dispositivo movil
offBtn = (Button) findViewById(R.id.turnOff);
offBtn.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        off(v);
    }
});
//dispositivos emparejados con anterioridad
listBtn = (Button) findViewById(R.id.paired);
listBtn.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        list(v);
    }
});
//busqueda de nuevos dispositivos
findBtn = (Button) findViewById(R.id.search);
findBtn.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        find(v);
    }
});

myListView = (ListView) findViewById(R.id.listView1);

//se muestran los dispositivos en la lista
BTArrayAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1);
myListView.setAdapter(BTArrayAdapter);
myListView.setOnItemClickListener(myListClickListener);
}
}
@Override
public boolean onSupportNavigateUp() { //función que permite
acceder a la pantalla anterior en la toolbar
    onBackPressed();
    return false;
}

void on(View view) { //comprobación del estado del bluetooth en
el dispositivo móvil
    if (!myBluetoothAdapter.isEnabled()) {
        Intent turnOnIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(turnOnIntent, REQUEST_ENABLE_BT);
text.setText("ESTADO: ACTIVADO");
    }
}

```

```

        Toast.makeText(getApplicationContext(), "BLUETOOTH
ACTIVADO",
            Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(getApplicationContext(), "BLUETOOTH YA ESTA
ACTIVADO",
            Toast.LENGTH_LONG).show();
        text.setText("ESTADO: ACTIVADO");
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
Intent data) { //se muestra aviso en pantalla del estado del bluetooth
del telefono
    // TODO Auto-generated method stub
    if (requestCode == REQUEST_ENABLE_BT) {
        if (myBluetoothAdapter.isEnabled()) {
            text.setText("ESTADO: ACTIVADO");
        } else {
            text.setText("ESTADO: DESACTIVADO");
        }
    }
}

public void list(View view) { //obtención de los dispositivos
emparejados
    pairedDevices = myBluetoothAdapter.getBondedDevices();
    for (BluetoothDevice device : pairedDevices)
        BTArrayAdapter.add(device.getName() + "\n" +
device.getAddress());
    Toast.makeText(getApplicationContext(), "MOSTRAR DISPOSITIVOS
EMPAREJADOS",
        Toast.LENGTH_SHORT).show();
}

final BroadcastReceiver bReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
//obtención de nuevos dispositivos
            BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            BTArrayAdapter.add(device.getName() + "\n" +
device.getAddress());
            BTArrayAdapter.notifyDataSetChanged();
        }
    }
};

public void find(View view) {
    if (myBluetoothAdapter.isDiscovering()) {
        // the button is pressed when it discovers, so cancel the
discovery
        myBluetoothAdapter.cancelDiscovery();
    } else {
        BTArrayAdapter.clear();
    }
}

```

```

        myBluetoothAdapter.startDiscovery();
        registerReceiver(bReceiver, new
IntentFilter(BluetoothDevice.ACTION_FOUND));
    }
}

public void off(View view) { //desactivación del bluetooth del
telefono móvil
    myBluetoothAdapter.disable();
    text.setText("ESTADO: DESACTIVADO");

    Toast.makeText(getApplicationContext(), "Bluetooth desactivado",
        Toast.LENGTH_LONG).show();
}

@Override
protected void onDestroy() {
    // TODO Auto-generated method stub
    super.onDestroy();
    unregisterReceiver(bReceiver);
}
//conexión y emparejamiento de dos dispositivos
private AdapterView.OnItemClickListener myListClickListener = new
AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView<?> av, View v, int arg2,
long arg3) {
        String info = ((TextView) v).getText().toString();
        String address = info.substring(info.length() - 17);
        Intent i = new Intent(Main4Activity.this,
MainActivity.class);

        EXTRA_ADDRESS=address;
        startActivity(i);
    }
};
private BluetoothSocket createBluetoothSocket(BluetoothDevice
device) throws IOException {

    return device.createRfcommSocketToServiceRecord(myUUID);
}

protected void onStop() {
    super.onStop();
    //Se obtiene la dirección MAC del dispositivo seleccionado en la
lista de dispositivos activos
    Intent newint = getIntent();
    //inicio del intento de conexión
    BluetoothDevice device =
myBluetoothAdapter.getRemoteDevice(EXTRA_ADDRESS);

    try {
        btSocket = createBluetoothSocket(device);
    } catch (IOException e) {
        Toast.makeText(getBaseContext(), "La creación del Socket
fallo", Toast.LENGTH_LONG).show();
    }

    try {

```



```

        btSocket.connect();
    } catch (IOException e) {
        try {
            btSocket.close();
        } catch (IOException e2) {

        }
    }
}

mConnectedThread = new ConnectedThread(btSocket);
mConnectedThread.start();
MainActivity.bluetoothemparejado=true; //emparejamiento del
bluetooth realizado
mConnectedThread.write("x");//enviamos una x para saber que la
vinculación se ha iniciado
}

public class ConnectedThread extends Thread { //creación de la
conexión entre los dos dispositivos
    private final InputStream mmInStream;
    private final OutputStream mmOutStream;

    public ConnectedThread(BluetoothSocket socket) {
        InputStream tmpIn = null;
        OutputStream tmpOut = null;

        try {
            //Create I/O streams for connection
            tmpIn = socket.getInputStream();
            tmpOut = socket.getOutputStream();
        } catch (IOException e) { }

        mmInStream = tmpIn;
        mmOutStream = tmpOut;
    }
    public void run() {
        byte[] buffer = new byte[256];
        int bytes;

        while (true) {
            try {
                bytes = mmInStream.read(buffer);
                String readMessage = new String(buffer, 0, bytes);

            } catch (IOException e) {
                break;
            }
        }
    }

    public void write(String input) {

        input=("AA"+input+"BB"); //generamos el paquete de
datos a enviar con el inicio y el final que debe recibir Arduino

        byte[] msgBuffer = input.getBytes(); //conversión
de los caracteres en String a bytes
        try {
            mmOutStream.write(msgBuffer); //envio el

```

```

paquete de bytes generados a través de bluetooth
    } catch (IOException e) {
        //en caso de perder la conexión mientras se están
enviando datos, se cierra la conexión
        Toast.makeText(getApplicationContext(), "La Conexión fallo",
Toast.LENGTH_LONG).show();
        finish();
    }
}
}
}
}

```

Teaching por JOINT

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">

    <TextView
        android:text="J1,J2,J3,J4"
        android:textAlignment="center"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:textSize="24sp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView14"
        android:layout_below="@+id/xyz_button"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="10dp"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <Spinner
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/listapuntos"
        android:layout_marginTop="11dp"
        android:textSize="14sp"
        android:layout_below="@+id/textView13"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_toLeftOf="@+id/guardarposicion"
        android:layout_toStartOf="@+id/guardarposicion" />

    <Button
        android:text="GUARDAR POSICIÓN"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/guardarposicion"
        android:layout_below="@+id/textView13"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

```

```

<Button
    android:text="JOINT"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/joint_button"
    android:textSize="18sp"
    android:layout_alignBaseline="@+id/xyz_button"
    android:layout_alignBottom="@+id/xyz_button"
    android:layout_alignLeft="@+id/guardarposicion"
    android:layout_alignStart="@+id/guardarposicion"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

<Button
    android:text="X Y Z"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/xyz_button"
    tools:textSize="18sp"
    android:layout_marginTop="60dp"
    android:layout_marginRight="56dp"
    android:layout_marginEnd="56dp"
    android:layout_alignParentTop="true"
    android:layout_toLeftOf="@+id/joint_button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/izquierda"
    android:id="@+id/ombro_izq"
    android:layout_below="@+id/cintura_izq"
    android:layout_alignLeft="@+id/cintura_izq"
    android:layout_alignStart="@+id/cintura_izq" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/izquierda"
    android:layout_below="@+id/ombro_izq"
    android:layout_alignLeft="@+id/ombro_izq"
    android:layout_alignStart="@+id/ombro_izq"
    android:id="@+id/codo_izq" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/izquierda"
    android:layout_below="@+id/codo_izq"
    android:layout_alignLeft="@+id/codo_izq"
    android:layout_alignStart="@+id/codo_izq"
    android:id="@+id/muneca_izq" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/derecha"
    android:layout_alignBottom="@+id/cintura_izq"

```

```

        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true"
        android:id="@+id/cintura_der" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/derecha"
    android:layout_below="@+id/cintura_der"
    android:layout_alignLeft="@+id/cintura_der"
    android:layout_alignStart="@+id/cintura_der"
    android:id="@+id/ombro_der" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/derecha"
    android:layout_above="@+id/muneca_izq"
    android:layout_alignLeft="@+id/ombro_der"
    android:layout_alignStart="@+id/ombro_der"
    android:id="@+id/codo_der" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/derecha"
    android:layout_alignBottom="@+id/muneca_izq"
    android:layout_alignLeft="@+id/codo_der"
    android:layout_alignStart="@+id/codo_der"
    android:id="@+id/muneca_der" />

<TextView
    android:text="HOMBRO"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView16"
    android:textSize="30sp"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_below="@+id/cintura_izq"
    android:layout_centerHorizontal="true" />

<TextView
    android:text="MUÑECA"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView22"
    android:textSize="30sp"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_below="@+id/codo_der"
    android:layout_alignLeft="@+id/textView16"
    android:layout_alignStart="@+id/textView16" />

<TextView
    android:text="Seleccione número de la posición:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView13"
    android:textAppearance="?android:attr/textAppearanceLarge"
    tools:textColor="@android:color/background_dark"
    android:textSize="18sp"

```

```

    android:layout_marginTop="19dp"
    android:layout_below="@+id/textView12"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

```

```

<TextView
    android:text="X, Y, Z"
    android:textAlignment="center"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textSize="24sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView12"
    android:layout_below="@+id/textView14"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="10dp"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

```

```

<TextView
    android:text="CODO"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView18"
    android:textSize="30sp"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_below="@+id/ombro_izq"
    android:layout_centerHorizontal="true" />

```

```

<TextView
    android:text="CINTURA"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView15"
    android:textSize="30sp"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_alignTop="@+id/cintura_izq"
    android:layout_centerHorizontal="true" />

```

```

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/seekBar2"
    android:layout_centerHorizontal="true"
    android:id="@+id/textView28" />

```

```

<include
    layout="@layout/toolbar_layout"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

```

```

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/izquierda"
    android:layout_marginLeft="14dp"

```



```

import android.widget.Button;
import android.widget.ImageButton;
import android.widget.SeekBar;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

public class Main6Activity extends AppCompatActivity {

    int punto=0;           //valor del punto inicial predeterminado en
    la lista
    Spinner listapuntos;  //definición lista puntos para guardar
    posiciones
    String[] puntos = {"SELECCIONAR", "PUNTO 1","PUNTO 2", "PUNTO 3",
    "PUNTO 4"}; //puntos dónde se pueden guardar posiciones
    String lectura; //paquete de datos que se enviará a traves de
    bluetooth

    String tipo_longitud2; //variable para el tipo de movimiento

    SeekBar seekBar2;
    TextView tv2;

    private Toolbar toolbar;

    Button joint_button;    //variable botón JOINT
    Button xyz_button;      //variable botón XYZ

    ImageButton cintura_izq; //variable para mover cintura
    ImageButton cintura_der; //variable para mover cintura
    ImageButton ombro_der;   //variable para mover hombro
    ImageButton ombro_izq;  //variable para mover hombro
    ImageButton codo_izq;   //variable para mover codo
    ImageButton codo_der;   //variable para mover codo
    ImageButton muneca_izq; //variable para mover muneca
    ImageButton muneca_der; //variable para mover muneca
    ImageButton mano_der;   //variable para mover mano
    ImageButton mano_izq;   //variable para mover mano

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main6);

        toolbar = (Toolbar)findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        seekBar2 = (SeekBar)findViewById(R.id.seekBar2);
        tv2 = (TextView)findViewById(R.id.textView28);
        seekBar2.setMax(2);
        seekBar2.setProgress(0);

        seekBar2.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener(){
            @Override
            public void onProgressChanged(SeekBar seekBar2, int i,
boolean b){ //función para definir la longitud del movimiento

```

```

        tv2.setText(i+"/"+seekBar2.getMax());
        if (i==0){
            tipo_longitud2="60"; //generamos y enviamos
paquete de datos para indicar movimiento corto
            Main4Activity.
mConnectedThread.write(tipo_longitud2);
        }
        if (i==1){
            tipo_longitud2="61"; //generamos y enviamos
paquete de datos para indicar movimiento medio
            Main4Activity.
mConnectedThread.write(tipo_longitud2);
        }
        if (i==2){
            tipo_longitud2="62"; //generamos y enviamos
paquete de datos para indicar movimiento largo
            Main4Activity.
mConnectedThread.write(tipo_longitud2);
        }
    }
    @Override
    public void onStartTrackingTouch(SeekBar seekBar){
    }
    @Override
    public void onStopTrackingTouch(SeekBar seekBar){
    }
});

Button guardarposicion;
guardarposicion = (Button) findViewById(R.id.guardarposicion);

listapuntos=(Spinner)findViewById(R.id.listapuntos);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, puntos);
listapuntos.setAdapter(adapter);

joint_button = (Button) findViewById(R.id.joint_button);
xyz_button = (Button) findViewById(R.id.xyz_button);

cintura_izq = (ImageButton) findViewById(R.id.cintura_izq);
//botón-imagen para mover cintura en un sentido
cintura_der = (ImageButton) findViewById(R.id.cintura_der);
//botón-imagen para mover cintura en otro sentido
ombro_der = (ImageButton) findViewById(R.id.ombro_der);
//botón-imagen para mover hombro en otro sentido
ombro_izq = (ImageButton) findViewById(R.id.ombro_izq); //botón-
imagen para mover hombro en un sentido
codo_der = (ImageButton) findViewById(R.id.codo_der); //botón-
imagen para mover codo en otro sentido
codo_izq = (ImageButton) findViewById(R.id.codo_izq); //botón-
imagen para mover codo en un sentido
muneca_der = (ImageButton) findViewById(R.id.muneca_der);
//botón-imagen para mover muneca en otro sentido
muneca_izq = (ImageButton) findViewById(R.id.muneca_izq);
//botón-imagen para mover muneca en un sentido
mano_izq= (ImageButton) findViewById(R.id.mano_izq); //botón-

```



```

imagen para mover mano en un sentido
    mano_der= (ImageButton) findViewById(R.id.mano_der);    //botón-
imagen para mover mano en otro sentido

    joint_button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {    //muestra la misma
pantalla (teaching por articulaciones)
            Intent joint_button = new Intent(Main6Activity.this,
Main6Activity.class);
            startActivity(joint_button);
        }
    });

    xyz_button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v){    //en caso de pulsar el
botón se accede a la pantalla de teaching por coordenadas
            Intent xyz_button = new Intent(Main6Activity.this,
Main5Activity.class);
            startActivity(xyz_button);
        }
    });

    cintura_izq.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v){    //generamos y enviamos
paquete de datos para mover la cintura en un sentido
            lectura="211";
            Main4Activity.mConnectedThread.write(lectura);
        }
    });

    cintura_der.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v){    //generamos y enviamos
paquete de datos para mover la cintura en otro sentido
            lectura="210";
            Main4Activity.mConnectedThread.write(lectura);
        }
    });

    ombro_izq.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v){    //generamos y enviamos
paquete de datos para mover el ombro en un sentido
            lectura="221";
            Main4Activity.mConnectedThread.write(lectura);
        }
    });

    ombro_der.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v){    //generamos y enviamos
paquete de datos para mover el ombro en otro sentido
            lectura="220";
            Main4Activity.mConnectedThread.write(lectura);
        }
    });

    codo_izq.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v){    //generamos y enviamos
paquete de datos para mover el codo en un sentido

```

```

        lectura="231";
        Main4Activity. mConnectedThread.write(lectura);
    }
});
codo_der.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v){           //generamos y enviamos
paquete de datos para mover el codo en otro sentido
        lectura="230";
        Main4Activity. mConnectedThread.write(lectura);
    }
});
muneca_izq.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v){           //generamos y enviamos
paquete de datos para mover la muñeca en un sentido
        lectura="241";
        Main4Activity. mConnectedThread.write(lectura);
    }
});
muneca_der.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v){           //generamos y enviamos
paquete de datos para mover la muñeca en otro sentido
        lectura="240";
        Main4Activity. mConnectedThread.write(lectura);
    }
});
mano_izq.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v){           //generamos y
enviamos paquete de datos para mover la mano en un sentido
        lectura="251";
        Main4Activity. mConnectedThread.write(lectura);
    }
});
mano_der.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v){           //generamos y
enviamos paquete de datos para mover la mano en otro sentido
        lectura="250";
        Main4Activity. mConnectedThread.write(lectura);
    }
});

listapuntos.setOnItemClickListener(new
AdapterView.OnItemClickListener() { //definición del valor del
punto en función de la selección del usuario
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
int i, long id) {
        switch (i){
            case 0:
                punto=0;
                break;
            case 1:
                punto=1;
                break;
            case 2:

```



```

    });
}
@Override
public boolean onSupportNavigateUp() { //función que permite
acceder a la pantalla anterior en la toolbar (en este caso conduce a la
pantalla principal)
    Intent inicio = new Intent(Main6Activity.this,
MainActivity.class);
    startActivity(inicio);
    onBackPressed();
    return false;
}
}
}

```

Teaching por X, Y, X:

```

<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:orientation="vertical"
android:layout_width="match_parent"
android:layout_height="match_parent">

<TextView
    android:text="J1,J2,J3,J4"
    android:textAlignment="center"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textSize="24sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView14"
    android:layout_below="@+id/xyz_button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="10dp"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

<Spinner
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/listapuntos"
    android:layout_marginTop="11dp"
    android:textSize="14sp"
    android:layout_below="@+id/textView13"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_toLeftOf="@+id/guardarposicion"
    android:layout_toStartOf="@+id/guardarposicion" />

<Button
    android:text="GUARDAR POSICIÓN"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/guardarposicion"

```

```

    android:layout_below="@+id/textView13"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

```

<Button

```

    android:text="JOINT"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/join_button"
    android:textSize="18sp"
    android:layout_alignBaseline="@+id/xyz_button"
    android:layout_alignBottom="@+id/xyz_button"
    android:layout_alignLeft="@+id/guardarposicion"
    android:layout_alignStart="@+id/guardarposicion"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

```

<Button

```

    android:text="X Y Z"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/xyz_button"
    tools:textSize="18sp"
    android:layout_marginTop="60dp"
    android:layout_marginRight="56dp"
    android:layout_marginEnd="56dp"
    android:layout_alignParentTop="true"
    android:layout_toLeftOf="@+id/join_button"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

```

<TextView

```

    android:text="EJE X"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/yas"
    android:layout_centerHorizontal="true"
    android:layout_marginBottom="27dp"
    android:id="@+id/textView19"
    android:textSize="30sp"
    android:textAppearance="?android:attr/textAppearanceLarge"/>

```

<TextView

```

    android:text="EJE Y"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/zas"
    android:layout_alignRight="@+id/textView19"
    android:layout_alignEnd="@+id/textView19"
    android:layout_marginBottom="24dp"
    android:id="@+id/textView20"
    android:textSize="30sp"
    android:textAppearance="?android:attr/textAppearanceLarge"/>

```

<TextView

```

    android:text="EJE Z"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/zas"
    android:layout_alignRight="@+id/textView20"

```

```

    android:layout_alignEnd="@+id/textView20"
    android:layout_marginBottom="24dp"
    android:id="@+id/textView21"
    android:textSize="30sp"
    android:textAppearance="?android:attr/textAppearanceLarge"/>

```

<TextView

```

    android:text="Seleccione número de la posición:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView13"
    android:textAppearance="?android:attr/textAppearanceLarge"
    tools:textColor="@android:color/background_dark"
    android:textSize="18sp"
    android:layout_marginTop="23dp"
    android:layout_below="@+id/textView12"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

```

<TextView

```

    android:text="X, Y, Z"
    android:textAlignment="center"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:textSize="24sp"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView12"
    android:layout_below="@+id/textView14"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginTop="10dp"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

```

<ImageButton

```

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/mas"
    android:id="@+id/zmas"
    android:layout_marginBottom="46dp"
    android:layout_alignParentBottom="true"
    android:layout_alignLeft="@+id/yas"
    android:layout_alignStart="@+id/yas" />

```

<ImageButton

```

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/mas"
    android:id="@+id/yas"
    android:layout_above="@+id/zmas"
    android:layout_alignLeft="@+id/xas"
    android:layout_alignStart="@+id/xas" />

```

<ImageButton

```

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/menys"
    android:id="@+id/zmenos"
    android:layout_below="@+id/textView20"
    android:layout_alignLeft="@+id/ymenos"

```

```

        android:layout_alignStart="@+id/ymenos" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/menys"
    android:id="@+id/ymenos"
    android:layout_above="@+id/zmenos"
    android:layout_alignLeft="@+id/xmenos"
    android:layout_alignStart="@+id/xmenos" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/mas"
    android:id="@+id/xmas"
    android:layout_marginLeft="21dp"
    android:layout_marginStart="21dp"
    android:layout_above="@+id/yas"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

<include layout="@layout/toolbar_layout"
    android:layout_alignParentTop="true"
    android:layout_alignRight="@+id/xmenos"
    android:layout_alignEnd="@+id/xmenos"
    android:layout_marginRight="42dp"
    android:layout_marginEnd="42dp" />

<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/menys"
    android:id="@+id/xmenos"
    android:layout_above="@+id/ymenos"
    android:layout_toRightOf="@+id/textView13"
    android:layout_toEndOf="@+id/textView13" />

<SeekBar
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView21"
    android:layout_centerHorizontal="true"
    android:id="@+id/seekBar" />

<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:id="@+id/textView27" />
</RelativeLayout>

package com.example.marcruizvinyeta.robotrichmar;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.view.View;

```

```

import android.widget.AdapterView;
import android.widget.AdapterView;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.SeekBar;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

public class Main5Activity extends AppCompatActivity {

    int punto=0;           //valor del punto inicial predeterminado en
    la lista
    String tipo_longitud; //variable para el tipo de movimiento
    Spinner listapuntos;  //definición lista puntos para guardar
    posiciones
    String[] puntos = {"SELECCIONAR", "PUNTO 1", "PUNTO 2", "PUNTO 3",
    "PUNTO 4"}; //puntos dónde se pueden guardar posiciones
    String lectura;      //paquete de datos que se enviará a traves de
    bluetooth

    SeekBar seekBar;
    TextView tv;

    private Toolbar toolbar;

    Button joint_button; //variable botón JOINT
    Button xyz_button;   //variable botón XYZ

    ImageButton xmas;   //variable para subir en X
    ImageButton xmenos; //variable para bajar en X
    ImageButton ymas;   //variable para subir en Y
    ImageButton ymenos; //variable para bajar en Y
    ImageButton zmas;   //variable para subir en Z
    ImageButton zmenos; //variable para bajar en Z

    @Override
    protected void onCreate(final Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main5);

        toolbar = (Toolbar)findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        seekBar = (SeekBar)findViewById(R.id.seekBar);
        tv = (TextView)findViewById(R.id.textView27);
        seekBar.setMax(2);
        seekBar.setProgress(0);

        seekBar.setOnSeekBarChangeListener(new
        SeekBar.OnSeekBarChangeListener(){ //función para definir la longitud
        del movimiento
            @Override
            public void onProgressChanged(SeekBar seekBar, int i,
boolean b){

```



```

        tv.setText(i+"/"+seekBar.getMax());
        if (i==0){
            tipo_longitud="60"; //generamos y enviamos
paquete de datos para indicar movimiento corto
            Main4Activity.
mConnectedThread.write(tipo_longitud);
        }
        if (i==1){
            tipo_longitud="61"; //generamos y enviamos
paquete de datos para indicar movimiento medio
            Main4Activity.
mConnectedThread.write(tipo_longitud);
        }
        if (i==2){
            tipo_longitud="62"; //generamos y enviamos
paquete de datos para indicar movimiento largo
            Main4Activity.
mConnectedThread.write(tipo_longitud);
        }
    }
    @Override
    public void onStartTrackingTouch(SeekBar seekBar){
    }
    @Override
    public void onStopTrackingTouch(SeekBar seekBar){
    }
});

Button guardarposicionejes;
guardarposicionejes = (Button)
findViewById(R.id.guardarposicion);

joint_button = (Button) findViewById(R.id.joint_button);
xyz_button = (Button) findViewById(R.id.xyz_button);

xmas = (ImageButton) findViewById(R.id.xmas); //botón-imagen
para subir a través del eje X
xmenos = (ImageButton) findViewById(R.id.xmenos); //botón-
imagen para bajar a través del eje X
ymas = (ImageButton) findViewById(R.id.ymas); //botón-imagen
para subir a través del eje Y
ymenos = (ImageButton) findViewById(R.id.ymenos); //botón-
imagen para bajar a través del eje Y
zmas = (ImageButton) findViewById(R.id.zmas); //botón-imagen
para subir a través del eje Z
zmenos = (ImageButton) findViewById(R.id.zmenos); //botón-
imagen para bajar a través del eje Z

joint_button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { //en caso de pulsar el botón
se accede a la pantalla de teaching por JOINT
        Intent joint_button = new Intent(Main5Activity.this,
Main6Activity.class);
        startActivity(joint_button);
    }
});

```

```

    });

    xyz_button.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { //muestra la misma pantalla
            (teaching por coordenadas)
            Intent xyz_button = new Intent(Main5Activity.this,
            Main5Activity.class);
            startActivity(xyz_button);
        }
    });

    xmas.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { //generamos y enviamos
            paquete de datos para subir a través del eje X
            lectura="111";
            Main4Activity.mConnectedThread.write(lectura);
        }
    });

    xmenos.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { //generamos y enviamos
            paquete de datos para bajar a través del eje X
            lectura="110";
            Main4Activity.mConnectedThread.write(lectura);
        }
    });

    ymas.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { //generamos y enviamos
            paquete de datos para subir a través del eje Y
            lectura="121";
            Main4Activity.mConnectedThread.write(lectura);
        }
    });

    ymenos.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { //generamos y enviamos
            paquete de datos para bajar a través del eje Y
            lectura="120";
            Main4Activity.mConnectedThread.write(lectura);
        }
    });

    zmas.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { //generamos y enviamos
            paquete de datos para subir a través del eje Z
            lectura="131";
            Main4Activity.mConnectedThread.write(lectura);
        }
    });

    zmenos.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) { //generamos y enviamos

```

```

paquete de datos para bajar a través del eje Z
        lectura="130";
        Main4Activity. mConnectedThread.write(lectura);
    }
});

listapuntos=(Spinner)findViewById(R.id.listapuntos);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, puntos);
listapuntos.setAdapter(adapter);

listapuntos.setOnItemClickListener(new
AdapterView.OnItemClickListener() { //definición del valor del
punto en función de la selección del usuario
    @Override
    public void onItemClick(AdapterView<?> parent, View view,
int i, long id) {
        switch (i){
            case 0:
                punto=0;
                break;
            case 1:
                punto=1;
                break;
            case 2:
                punto=2;
                break;
            case 3:
                punto=3;
                break;
            case 4:
                punto=4;
                break;
        }
    }
    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        punto=0;
    }
});
guardarposicionejes.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View v) { //función para guardar
posiciones en puntos 1, 2, 3 y 4
        if (punto == 1) {
            Toast.makeText(getApplicationContext(), "POSICION
GUARDADA EN PUNTO 1", Toast.LENGTH_LONG).show();
            lectura="312";
            Main4Activity. mConnectedThread.write(lectura);
        } else {
            if (punto == 2) {
                Toast.makeText(getApplicationContext(),
"POSICION GUARDADA EN PUNTO 2", Toast.LENGTH_LONG).show();
                lectura="322";
                Main4Activity. mConnectedThread.write(lectura);
            } else {

```



```

        android:layout_alignStart="@+id/textView23"
        android:layout_marginTop="29dp"
        android:id="@+id/textView24"
        android:textAppearance="?android:attr/textAppearanceLarge"/>

<TextView
    android:text="Punto 4:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/textView24"
    android:layout_alignLeft="@+id/textView24"
    android:layout_alignStart="@+id/textView24"
    android:layout_marginTop="29dp"
    android:id="@+id/textView25"
    android:textAppearance="?android:attr/textAppearanceLarge"/>

<Button
    android:text="INICIAR RECORRIDO"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="47dp"
    android:id="@+id/inicio"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true" />

<TextView
    android:text="Punto 2:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
    android:id="@+id/textView23"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_below="@+id/textView10"
    android:layout_alignLeft="@+id/textView10"
    android:layout_alignStart="@+id/textView10" />

<TextView
    android:text="Punto 1:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView10"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_marginTop="128dp"
    android:layout_marginRight="15dp"
    android:layout_marginEnd="15dp"
    android:layout_alignParentTop="true"
    android:layout_toLeftOf="@+id/inicio"
    android:layout_toStartOf="@+id/inicio" />

<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="textPersonName"
    android:text="Ejemplo:1,2,3,4"
    android:ems="10"
    android:id="@+id/puntosrecorrido"
    android:layout_below="@+id/textView26"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="18dp"

```

```

        tools:textAlignment="center" />

<include layout="@layout/toolbar_layout"
    android:layout_alignParentTop="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true" />

<Switch
    android:text="Pinza"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/pinzapcuatro"
    android:textOff="CERRADA"
    android:textOn="ABIERTA"
    android:layout_alignBottom="@+id/textView25"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
<Switch
    android:text="Pinza"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/pinzapuno"
    android:textOff="CERRADA"
    android:textOn="ABIERTA"
    android:layout_alignBottom="@+id/textView10"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
<Switch
    android:text="Pinza"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/pinzapdos"
    android:textOff="CERRADA"
    android:textOn="ABIERTA"
    android:layout_alignBottom="@+id/textView23"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />
<Switch
    android:text="Pinza"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/pinzaptres"
    android:textOff="CERRADA"
    android:textOn="ABIERTA"
    android:layout_alignBottom="@+id/textView24"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

<TextView
    android:text="Escriba aquí la secuencia de recorrido: (sólo 4
puntos)"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:id="@+id/textView26"
    tools:textSize="14sp"
    android:layout_below="@+id/textView25"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="26dp" />

```

```

<TextView
    android:text="RECORRIDO"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/recorrido"
    android:textAlignment="center"
    android:layout_below="@+id/puntosrecorrido"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />

</RelativeLayout>

package com.example.maracruzvinjeta.robotrichmar;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.widget.Toolbar;
import android.view.View;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.EditText;
import android.widget.Switch;
import android.widget.TextView;

public class Main7Activity extends AppCompatActivity {

    private Toolbar toolbar;
    Switch pinzapuno; //boton switch para posición de la pinza en
el punto 1
    Switch pinzapdos; //boton switch para posición de la pinza en
el punto 2
    Switch pinzaptres; //boton switch para posición de la pinza en
el punto 3
    Switch pinzapcuatro; //boton switch para posición de la pinza en
el punto 4

    String codigo; //paquete que indica el recorrido a realizar
    String codigo_pinza; //paquete que indica las posiciones de la
pinza en cada punto

    int pinza1=1; //estado inicial de la pinza en el punto 1
    int pinza2=1; //estado inicial de la pinza en el punto 2
    int pinza3=1; //estado inicial de la pinza en el punto 3
    int pinza4=1; //estado inicial de la pinza en el punto 4

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main7);

        toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        Button inicio;
        final TextView recorrido = (TextView)

```

```

findViewById(R.id.recorrido);

inicio = (Button) findViewById(R.id.inicio);

pinzapuno = (Switch) findViewById(R.id.pinzapuno);
pinzapdos = (Switch) findViewById(R.id.pinzapdos);
pinzaptres = (Switch) findViewById(R.id.pinzaptres);
pinzapcuatro = (Switch) findViewById(R.id.pinzapcuatro);

pinzapuno.setChecked(true);
pinzapdos.setChecked(true);
pinzaptres.setChecked(true);
pinzapcuatro.setChecked(true);

pinzapuno.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) { //asignacion del estado de la pinza en el punto
1
        if(isChecked){
            pinza1=1;
            codigo_pinza=("5"+pinza1+pinza2+pinza3+pinza4);
            Main4Activity.mConnectedThread.write(codigo_pinza);
        }else{
            pinza1=0;
            codigo_pinza=("5"+pinza1+pinza2+pinza3+pinza4);
            Main4Activity.mConnectedThread.write(codigo_pinza);
        }
    }
});
pinzapdos.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) { //asignacion del estado de la pinza en el punto
2
        if(isChecked){
            pinza2=1;
            codigo_pinza=("5"+pinza1+pinza2+pinza3+pinza4);
            Main4Activity.mConnectedThread.write(codigo_pinza);
        }else{
            pinza2=0;
            codigo_pinza=("5"+pinza1+pinza2+pinza3+pinza4);
            Main4Activity.mConnectedThread.write(codigo_pinza);
        }
    }
});
pinzaptres.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) { //asignacion del estado de la pinza en el punto
3
        if(isChecked){
            pinza3=1;
            codigo_pinza=("5"+pinza1+pinza2+pinza3+pinza4);
            Main4Activity.mConnectedThread.write(codigo_pinza);
        }else{
            pinza3=0;

```



```

        codigo_pinza= ("5"+pinza1+pinza2+pinza3+pinza4);
        Main4Activity. mConnectedThread.write(codigo_pinza);
    }
}
});
pinzapcuatro.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) { //asignacion del estado de la pinza en el punto
4
        if(isChecked){
            pinza4=1;
            codigo_pinza= ("5"+pinza1+pinza2+pinza3+pinza4);
            Main4Activity. mConnectedThread.write(codigo_pinza);
        }else{
            pinza4=0;
            codigo_pinza= ("5"+pinza1+pinza2+pinza3+pinza4);
            Main4Activity. mConnectedThread.write(codigo_pinza);
        }
    }
});

inicio.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) { //tratamiento y verificación
de los datos que introduce el usuario para realizar un recorrido

        EditText puntosrecorrido = (EditText)
findViewById(R.id.puntosrecorrido);
        String lectura;

        lectura = puntosrecorrido.getText().toString();

        int i;

        String[] separated = lectura.split(",", 100);
        i = separated.length;

        if (i > 4) {
            recorrido.setText("SOBRAN PUNTOS");
        } else {
            if (i < 4){
                recorrido.setText("FALTAN PUNTOS");
            }else{
                if ((separated[0].isEmpty()) ||
(separated[1].isEmpty()) || (separated[2].isEmpty()) ||
(separated[3].isEmpty())) {
                    recorrido.setText("DATO VACÍO");
                } else {
                    char[] PUNO, PDOS, PTRES, PCUATRO;
                    PUNO = separated[0].toCharArray();
                    PDOS = separated[1].toCharArray();
                    PTRES = separated[2].toCharArray();
                    PCUATRO = separated[3].toCharArray();

```

```

        if ((Character.isDigit(PUNO[0])) &&
(Character.isDigit(PDOS[0])) && (Character.isDigit(PTRES[0])) &&
(Character.isDigit(PCUATRO[0]))) {
            int numuno =
Integer.parseInt(String.valueOf(PUNO[0]));
            int numdos =
Integer.parseInt(String.valueOf(PDOS[0]));
            int numtres =
Integer.parseInt(String.valueOf(PTRES[0]));
            int numcuatro =
Integer.parseInt(String.valueOf(PCUATRO[0]));

            if
((numuno<5)&&(numdos<5)&&(numtres<5)&&(numcuatro<5)) {
                if (numuno==0){
                    recorrido.setText("RECORRIDO NO
VÁLIDO");
                }
                if(numdos==0){
                    if
((numtres==0)&&(numcuatro==0)){
                        recorrido.setText("RECORRIDO
VÁLIDO");
                    }
                    codigo=("4"+numuno+numdos+numtres+numcuatro);
                    Main4Activity.
mConnectedThread.write(codigo);
                }else{
                    recorrido.setText("RECORRIDO
NO VÁLIDO");
                }
            }else{
                if(numtres==0){
                    if (numcuatro==0){
                        recorrido.setText("RECORRIDO VÁLIDO");
                    }
                    codigo=("4"+numuno+numdos+numtres+numcuatro);
                    Main4Activity.
mConnectedThread.write(codigo);
                }else{
                    recorrido.setText("RECORRIDO NO VÁLIDO");
                }
            }
        }else{
            if (numcuatro==0){
                recorrido.setText("RECORRIDO VÁLIDO");
            }
            codigo=("4"+numuno+numdos+numtres+numcuatro);
            Main4Activity.
mConnectedThread.write(codigo);
        }else{
            recorrido.setText("RECORRIDO VÁLIDO");
            codigo=("4"+numuno+numdos+numtres+numcuatro);
            Main4Activity.
mConnectedThread.write(codigo);
        }else{
            recorrido.setText("RECORRIDO VÁLIDO");
            codigo=("4"+numuno+numdos+numtres+numcuatro);
            Main4Activity.
mConnectedThread.write(codigo);
        }
    }
}

```

```

Main4Activity.
mConnectedThread.write(codigo);

        }
    }
} else {
    recorrido.setText("HAY PUNTOS NO
EXISTENTES");
}
} else {
    recorrido.setText("LETRAS NO VÁLIDAS");
}
}
}
}
});
@Override
public boolean onSupportNavigateUp() { //función que permite
acceder a la pantalla anterior en la toolbar
    onBackPressed();
    return false;
}
}
}

```

