UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO E IMPLEMENTACIÓN DEL CONTROL REMOTO DE UN BRAZO ROBÓTICO EDUCACIONAL UTILIZANDO DISPOSITIVOS BASADOS EN SISTEMA OPERATIVO ANDROID

TRABAJO FINAL DEL

Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR

Daniel Uroz Franco

TUTORIZADO POR

Ricardo Pizá Fernández

FECHA: **Valencia, junio, 2019**

RESUMEN

El proyecto trata el diseño, el desarrollo y la implementación de una aplicación para dispositivo móviles basada en *Android* y de un programa para la plataforma *Arduino*. Todo esto con el fin de manejar un brazo robótico educativo, incluido en el kit *Arm Robot* de la marca *Ebotics*. La comunicación entre los distintos programas se ha llevado a cabo vía *bluetooth* mediante un módulo extra añadido basado en *Arduino*.

Para ello, primero se ha realizado una breve introducción acerca de los avances tecnológicos en los últimos años, así como la facilidad a la hora de acceder a ellos y de aprender acerca de éstos, además de la motivación que ha ayudado a la realización del proyecto.

A continuación, se han comentado los diferentes elementos tanto *hardware* como *software* que han sido necesarios durante el desarrollo del proyecto y se han explicado algunas de sus características principales, así como algunos detalles acerca de éstos a tener en cuenta.

Por último, se ha explicado todo el desarrollo llevado a cabo para la realización del proyecto, desde el montaje del brazo robótico y el direccionamiento de los pines de la placa, hasta la configuración necesaria de los distintos elementos *hardware* que forman parte del proyecto y toda la programación diseñada en ambas plataformas, en *Android* y *Arduino*.

Palabras clave: aplicación, *Android*, programa, *Arduino*, *Arm Robot*, *bluetooth*.

ABSTRACT

The project addresses the design, development and implementation of an *Android*-based application for mobile devices and a program for the *Arduino* platform. All that in order to manipulate an educational robotic arm, included in the *Arm Robot* kit of the *Ebotics* brand. The communication between the different programs has been carried out via *bluetooth* by an extra module added based on *Arduino*.

For this effect, first a brief introduction has been made about the technological advances in the recent years, as well as the ease of accessing them and learning about them, besides the motivation that has helped about the realization of the project.

Next, the different *hardware* and *software* elements that have been necessary during the development of the project have been mentioned and some of their main characteristics have been explained, as well as some details about these to be taken in mind.

Finally, all the development carried out for the realization of the project has been explained, including the assembly of the robotic arm and the addressing of the pins of the board, the configuration that have been necessary in the different *hardware* elements that take part of the project and all the programming designed in both platforms, in *Android* and *Arduino*.

Keywords: application, *Android*, program, *Arduino*, *Arm Robot*, *bluetooth*.

CONTENIDOS

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO E IMPLEMENTACIÓN DEL CONTROL REMOTO DE UN BRAZO ROBÓTICO EDUCACIONAL UTILIZANDO DISPOSITIVOS BASADOS EN SISTEMA OPERATIVO ANDROID

DOCUMENTO 1.-

MEMORIA DEL PROYECTO

REALIZADO POR

Daniel Uroz Franco

TUTORIZADO POR

Ricardo Pizá Fernández

FECHA: **Valencia, junio, 2019**

INDICE

1. Objetivo

El objetivo del proyecto es el desarrollo de una aplicación móvil mediante el programa *Android Studio*, de código abierto, que sirva de interfaz para el manejo de un brazo robótico educativo, controlado mediante *Arduino*.

La aplicación consiste en una selección de modos de funcionamiento, entre los que se encuentran el tipo de control, así como otros tipos de funcionalidad distintas.

Los tipos de control que se han diseñado son dos: remoto y de periferia. El primero consiste en el manejo del robot por *bluetooth* mediante un dispositivo móvil que esté basado en el lenguaje de programación *Android*, mientras que el de periferia hace uso de unas entradas físicas conectadas a unas palancas de mando o *joysticks analógicos* que permiten su control.

2. Introducción

A lo largo del proyecto se plantea un posible desarrollo tanto de la aplicación móvil basada en *Arduino* que sirve como interfaz para el usuario como de la programación de la placa electrónica de *Arduino* que se encarga de los movimientos del brazo robótico.

2.1. ANTECEDENTES

Cada vez más se está introduciendo lentamente en nuestras vidas, por ejemplo, como parte de la formación educativa de los jóvenes, el mundo de los microprocesadores y cómo se estructura su lenguaje de programación. Esto está siendo posible gracias a que la disponibilidad de éstos está siendo cada vez mayor, además de su sencillez a la hora de realizar pequeños proyectos. lo mismo que le ocurre a la programación de las aplicaciones móviles.

Esto mismo les ocurre a las aplicaciones móviles, las cuales ya forman parte de nuestra vida cotidiana, ya sea para enviar o recibir mensaje, para facilitarnos diversas acciones o simplemente como entretenimiento, además de poder ofrecernos otros servicios.

El crecimiento que han sufrido estos tipos de lenguajes de programación de código abierto es debido en parte a eso mismo, a que cualquier usuario es capaz de aprender a entender sus estructuras y a manejarlos. Esto es gracias a que el *software* necesario para poder utilizarlos se encuentra de forma gratuita y a que en sus páginas web correspondientes también hay a disposición de cualquier usuario registrado numerosos tutoriales y manuales.

Gracias a esto y con la ayuda de Internet, se ha propiciado la creación de comunidades enteras de programadores autodidactas que ha favorecido aún más el crecimiento de este tipo de lenguajes de programación, así como el desarrollo de versiones mejores y de nuevas herramientas que nos faciliten nuestro día a día.

2.2. MOTIVACIÓN

Adquirir unos conocimientos básicos sobre algún ámbito no tan conocido pero que de alguna forma se pudiera relacionar con alguna de las materias estudiadas durante el Grado de Ingeniería Electrónica Industrial y Automática es uno de los factores por los que se ha optado para la realización del proyecto.

Otro de los factores que han influido en esta decisión ha sido el interés por la programación de aplicaciones móviles y en cómo se diferencia respecto a los lenguajes de programación estudiados durante el grado.

Por último, añadir que la comodidad de poder trabajar desde casa gracias a que el diseño del brazo robótico educativo permite su fácil transporte y manejo también ha contribuido a decantarse por el proyecto.

3. Solución adoptada

3.1. HARDWARE

3.1.1. KIT DE ROBÓTICA, ELECTRÓNICA Y PROGRAMACIÓN

De entre todo el catálogo de robots educativos existentes en el mercado actual, se ha optado por el kit de robótica, electrónica y programación basado en el concepto *Do It Yourself* (DIY) llamado *Arm Robot,* de la marca *Ebotics*, debido a su precio asequible para las capacidades de las que dispone, además de su apariencia similar con los brazos robóticos que se pueden encontrar en las industrias.

Gracias a este kit se puede construir un pequeño brazo robot educativo con 4 grados de libertad, capaz de levantar objetos ligeros de hasta 9g. Su estructura está constituida por piezas de metacrilato de color negro. Además, es compatible con la plataforma *Arduino* debido a que su placa *Build&Core UNO* contiene un microprocesador de la misma familia que la plataforma, de forma que su programación resulta sencilla.

Figura 1. Kit de robótica y electrónica Arm Robot de Ebotics

Este kit incluye los siguientes elementos:

- 1 placa *Build&Code UNO R3*
- 2 módulos de *joystick*
- 1 *Ebotics Sensor-Shieldv5.0*
- 4 servomotores *Microservo SG90 9g*
- 3 cables puente macho-hembra de 30cm.
- 8 cables puente hembra-hembra de 30cm.
- 1 estructura para brazo de placas troqueladas.
- 1 placa para mando
- 1 cable USB tipo A – USB tipo B
- 4 tornillos M3*25
- 18 tornillos M3*10
- 18 tornillos M3*8
- 7 tornillos M3*6
- 5 tornillos M2*5
- 2 tuercas M3
- 1 clavija de batería 9v

Incluye además un manual en diferentes idiomas con instrucciones paso a paso para poder realizar su montaje de forma sencilla, así como unos ejercicios prácticos para aprender a entender este tipo de lenguajes de programación y un apartado de preguntas frecuentes donde resolver algunas de las dudas más comunes.

3.1.2. PLACA BUILD&CORE UNO R3

La placa *Build&Core UNO R3* es una placa muy cómoda de usar a la hora de introducirse en el mundo de la electrónica y la programación debido a que utiliza *Arduino,* una plataforma de código abierto que permite a cualquier usuario ser autodidacta en este ámbito.

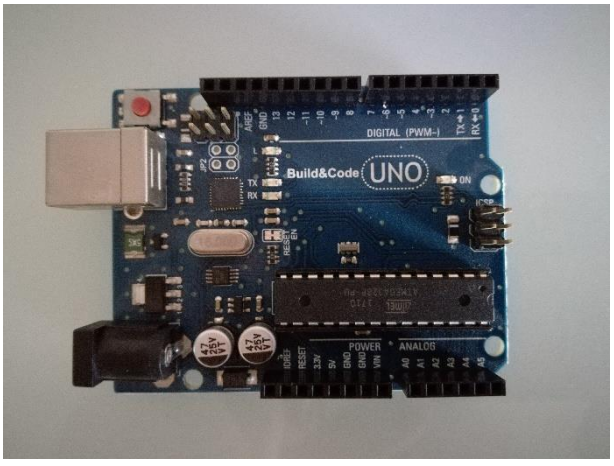

Figura 2. Placa Build&Core UNO R3

Las especificaciones técnicas de la placa son las siguientes:

- Microcontrolador *ATmega328P*

- Voltaje de funcionamiento: 5V
- Voltaje de entrada: 7-12V (recomendado), 6-20V (límite)
- 14 pines entrada/salida (I/O) digitales (de los cuales 6 suministran salida PWM)
- 6 pines entrada (inputs) analógicos
- Corriente por pin I/O: 20mA
- Corriente por pin 3.3V: 50mA
- Velocidad de reloj: 16MHz
- Tamaño: 68 x 53mm
- Peso: 25g
- Programable con *Arduino IDE* y *mBlock* (*Scratch* 2.0)
- Compatible con Mac OS, Windows y Linux

Como cualquier placa electrónica, en su diagrama de pines se puede observar cómo están distribuidos cada uno de éstos y su funcionalidad, de forma que se pueden deducir las limitaciones de la placa.

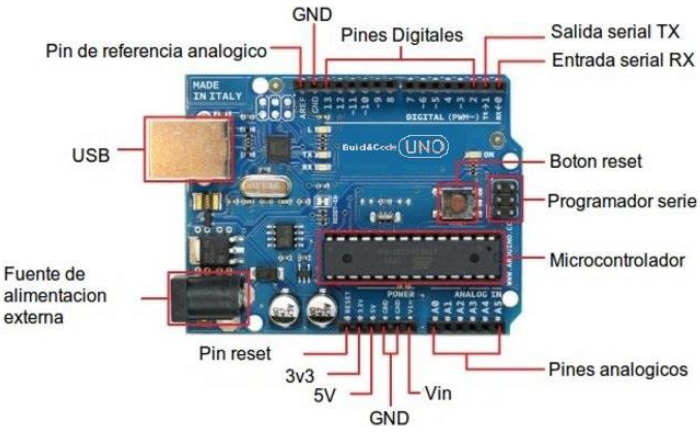

Figura 3. Diagrama de pines placa Build&Core UNO R3

Esta placa dispone de varias formas de alimentación: mediante un cable *USB* conectándolo a un ordenador, que además serviría para poder programarlo, o conectándolo a una fuente externa gracias a que cuenta con un conector *Jack* de 2.1mm para conectar un adaptador que se encuentre entre la tensión recomendada.

A continuación, se explicará de forma breve la funcionalidad de cada uno de estos pines:

- **VIN:** Mediante este pin obtenemos la alimentación a la que esté sometida la placa, regulada o no dependiendo de cómo esté alimentada
- **GND:** Conexión a masa
- **5V:** Se puede utilizar tanto para alimentar la placa si la fuente externa está regulada a 5V o, si ya está la placa alimentada tanto por *Jack* como por *USB,* puede servir para alimentar otro componente con una tensión regulada de 5V
- **3v3:** Este pin es utilizado para alimentar otros dispositivos a 3.3V. Se alimenta gracias al conector *Jack* o al cable *USB*

- **Pines analógicos:** La placa cuenta con 6 pines de entrada analógicas cuya resolución es de 10 bits. Miden tensiones de entre 0 y 5V, aunque es posible la modificación de este rango gracias a la función del pin AREF
- **Pin de referencia analógico:** Ofrece un voltaje de referencia para las entradas analógicas
- **Pin reset:** Tiene el mismo funcionamiento que el botón *reset* incorporado en la placa, cuya utilidad es reiniciar el microcontrolador
- **Pines digitales:** Componen un total de 14 pines de entradas/salidas digitales configurables desde la programación. Ofrecen una tensión de 5V
- **Salida serial Tx/Entrada serial Rx:** Se utilizan para recibir y transmitir datos en serie

3.1.3. EBOTICS SENSOR-SHIELDV5.0

Ebotics Sensor-Shieldv5.0 es una placa de expansión de pines aplicable a una variada cantidad de placas electrónicas basadas en *Arduino*. Es de gran utilidad a la hora de realizar las conexiones con otros sensores o actuadores, como los servomotores en nuestro caso, ya que éstas se realizan de forma muy sencilla y sin necesidad de utilizar tarjetas de expansión para soldar los componentes. Esta placa, además, también mantiene los mismos pines que la placa electrónica *Build&Core UNO R3,* por lo que no varía con respecto a la programación.

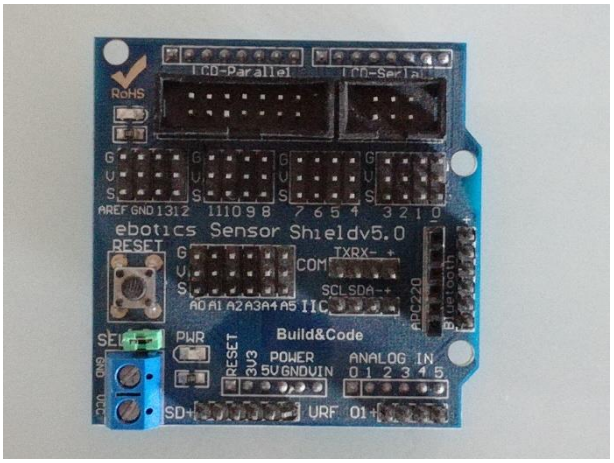

Figura 4. Ebotics Sensor-Shieldv5.0

Las características de esta placa de expansión son las siguientes:

- Voltaje de operación: 5V
- Compatible con *Arduino UNO, MEGA, LEONARDO, DUE*
- 14 pines entrada/salida (I/O) digitales
- 6 pines de entrada analógica
- Puerto *UART*
- Puerto *I2C*
- Puerto para módulo inalámbrico *APC220*
- Puerto para módulo *bluetooth HC06* o *HC05*
- Puerto para módulo *SD card*
- Puerto para *URF01+*
- Puerto para *LCD* paralelo

- Puerto para *LCD* serial
- *Led Power* y *Led L* (Pin 13)
- Pulsador *Reset*
- Bornera para alimentación externa

Con el diagrama de pines se puede observar la distribución de sus pines, factor a tener en cuenta a la hora de realizar el cableado de cualquier proyecto.

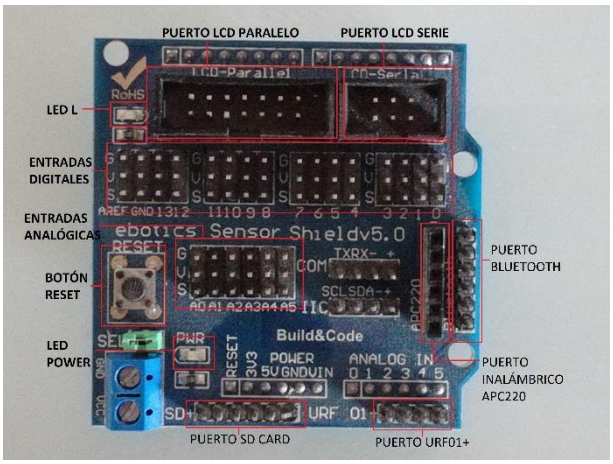

Figura 5. Diagrama de pines Ebotics Sensor-Shieldv5.0

3.1.4. MÓDULO BLUETOOTH HC-05

Para la conexión *bluetooth* con el dispositivo móvil, se ha incorporado un módulo *bluetooth HC-05* de *Arduino*, no estando éste incluido en el kit de *Ebotics.* Al finalizar su configuración, este módulo nos permitirá comunicarnos con nuestro dispositivo móvil para poder controlar el brazo robótico.

El módulo *bluetooth HC-05* es el dispositivo que permitirá la conexión inalámbrica entre la placa electrónica *Build&Core UNO R3* y el dispositivo móvil. Este módulo resulta muy sencillo a la hora de realizar su configuración y muy económico, de ahí se elección para realizar esta función.

Figura 6. Módulo bluetooth HC-05

Existe otro modelo, el módulo *bluetooth HC-06,* que compite muy a la par con el seleccionado, pero, debido a que el seleccionado dispone de configuraciones más avanzadas y que tiene más modos de funcionamiento, por lo que permite más funcionalidad y un establecimiento de la comunicación *bluetooth* mejor, se ha prescindido del modelo *HC-06*.

El modelo *HC-05* dispone de seis pines, a diferencia del otro modelo mencionado que dispone de cuatro. Esto es debido a que, como ya se ha comentado antes, tienen importantes diferencias de funcionalidad y de manejo, siendo un ejemplo de esto el modo de funcionamiento: el modelo *HC-06* funciona solamente como esclavo mientras que el modelo seleccionado puede actuar tanto como maestro como esclavo en la comunicación.

En esos seis pines de los que dispone se encuentra:

- **STATE:** Refleja el modo en el que se encuentra el módulo
- **RXD:** Patilla encargada de recibir los datos de la placa electrónica para enviarlos al dispositivo conectado vía *bluetooth*
- **TXD:** Patilla encargada de transmitir los datos, recibidos por el dispositivo al que se encuentre conectado vía *bluetooth,* a la placa electrónica
- **GND:** Conexión a masa
- **VCC:** Patilla de alimentación
- **EN:** Permite entrar al módulo en un modo de funcionamiento específico

3.1.5. SERVOMOTORES MICROSERVO SG90 9G

Los servomotores *Microservo SG90 9g* son ideales para prácticas de electrónica y robótica básicas, además de tener un importante papel en la educación debido a su bajo consumo, su reducido tamaño y a que resultan muy económicos.

Figura 7. Servomotor Microservo SG90 9g

Estos servomotores disponen de tres cables diseñados como una unidad de forma que facilita su conexión a placas electrónicas. Estos tres cables corresponden a la alimentación de los motores (cable rojo), la masa (cable marrón) y el cable correspondiente a la señal digital que le llega a éste para controlar su giro (cable naranja).

Las características de este servomotor son las siguientes:

- Dimensiones (L x W x H): 22.0 x 11.5 x 27mm
- Peso: 9g
- Peso con cable y conector: 10.6g
- Par a 4.8V: 16.7oz/in o 1.2kg/cm
- Voltaje de operación: 4.0 a 7.2V
- Velocidad de giro a 4.8V: 0.12seg/60º
- Conector universal para la mayoría de los receptores de radio control
- Compatible con tarjetas como *Arduino* y microcontroladores que funcionan a 5V

Otra característica de estos servomotores es que su modo de funcionamiento es mediante señales PWM. Esto es que funcionan mediante pulsos digitales enviados a 50Hz, es decir, cada 20ms. La anchura de la señal de este pulso es lo que determina el ángulo de giro del servomotor, que normalmente varía entre 0.5 y 2.5ms, (0º y 180º respectivamente).

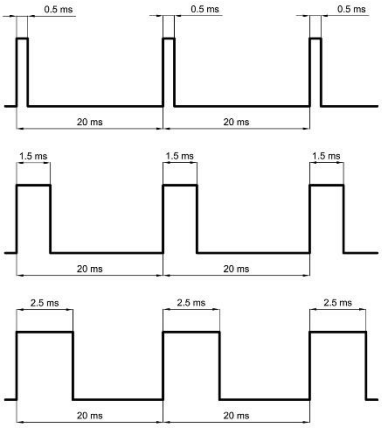

Figura 8. Señales PWM

Existen dos tipos de servomotores con respecto a la amplitud del giro que permiten: con tope mecánico y sin tope mecánico. El primero cuenta con un tope físico por el cual puede realizar un giro de 180º como máximo. Por otra parte, el servomotor sin tope mecánico puede realizar rotaciones completas debido a que carece del tope físico que caracteriza al anterior.

Para el proyecto disponemos de servomotores sin tope mecánico, de forma que se pueden permitir giros más amplios, pero debido al diseño de la estructura del brazo, ésta será la que limite los movimientos.

3.1.6. JOYSTICK ANALÓGICO

Un *joystick* analógico es un dispositivo que consiste en una palanca sujetada por un sistema de balancín con dos ejes ortogonales acoplados a dos potenciómetros, los cuales realizan la medición de la posición de la palanca en ambos ejes. Este sistema permite una amplia maniobrabilidad, por lo que es muy utilizado en sistemas que necesiten de un manejo más amplio, como robots.

Estos ejes están a su vez apoyados sobre un pulsador por lo que, además de enviar señales analógicas indicando la posición en la que se encuentra la palanca tanto en el eje x como en el eje y, también es capaz de enviar una señal digital indicando si el pulsador está accionado o no.

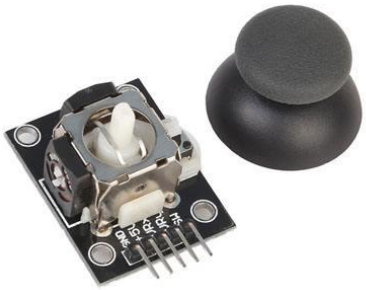

Figura 9. Joystick analógico

Los pines de los que disponen este tipo de módulos con cinco:

- **GND:** Conexión a masa
- **+5V:** Alimentación
- **VRX:** Valor analógico de la posición en el eje x
- **VRY:** Valor analógico de la posición en el eje y
- **SW:** Valor digital del pulsador

3.1.7. DISPOSITIVO MÓVIL HUAWEI P9 LITE

Con respecto al dispositivo móvil utilizado para la instalación de la aplicación, se ha optado por un teléfono móvil de la marca *Huawei* con modelo *P9 lite.* La decisión a la hora de escoger este modelo en concreto se basa principalmente en que su sistema operativo está basado en *Android*, requisito básico ya que la programación de la aplicación se ha realizado para dispositivos con esa base, y en la disponibilidad de éste, debido a que corresponde al teléfono móvil personal.

Figura 10. Huawei P9 Lite

Las especificaciones técnicas de este modelo son las siguientes:

- Pantalla IPS de 5.2'' 1080 x 1920 píxeles
- Procesador Kirin 650 de ocho núcleos
- Memoria RAM de 3GB
- Almacenamiento interno de 16GB

- Disponibilidad de tarjeta MicroSD de hasta 128GB
- Cámara principal Sony IMX214 de 13MP
- Cámara frontal de 8MP f/2.0
- Batería no extraíble de 3000mAh
- Lector de huellas
- NFC/GPS
- Conexión MicroUSB 2.0
- Tamaño de 146.8 x 72.6 x 7.5mm
- Peso de 147g
- Sistema operativo EMUI versión 5.0.3
- Versión de Android 7.0

3.2. SOFTWARE

3.2.1. ARDUINO IDE

El programa utilizado para la programación de la placa electrónica incorporada en el brazo robótico educativo es *Arduino IDE* versión 1.8.9, perteneciente a la misma marca del microprocesador, debido a su sencillez a la hora de utilizarlo. Éste es un programa gratuito escrito en *Java* y basado en *Processing* y otros lenguajes de programación de código abierto, siendo C++ el utilizado para el desarrollo de los programas, de forma que es ideal para usuarios autodidactas que quieran desarrollar sus propios proyectos.

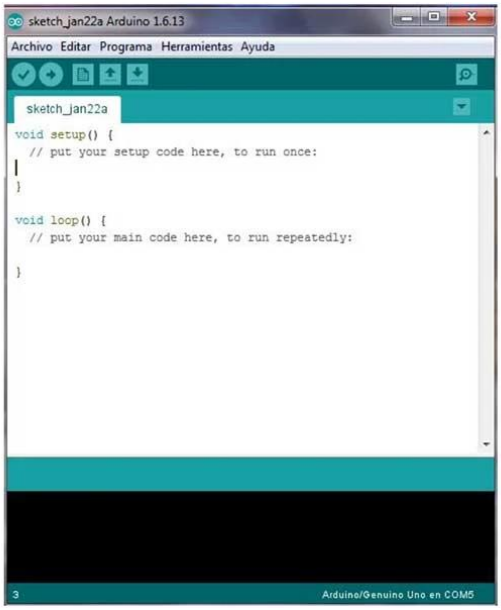

Figura 11. Interfaz Aruidno IDE

La interfaz que presenta este programa es muy sencilla, lo que facilita trabajar en proyectos que requieran de esta aplicación. Como en cualquier aplicación de escritorio, en la parte superior de ésta se encuentra el nombre del proyecto que esté abierto, además de las típicas opciones reunidas en varios menús desplegables.

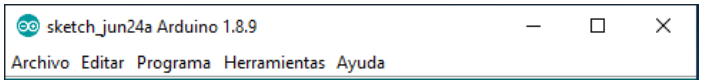

Figura 12. Menús desplegables de la interfaz Arduino IDE

A continuación, se encuentran una serie de atajos a las opciones más básicas y más utilizadas de esta aplicación. Siguiendo el orden de izquierda a derecha son:

- **Verificar:** Compila el código escrito en el programa para detectar posibles errores de programación.
- **Subir:** Carga el código en la placa conectada mediante conexión *USB.* Si no está compilado el código, esta función lo hace antes de cargarla en la placa.
- **Nuevo:** Abre un nuevo proyecto vacío de *Arduino IDE.*
- **Abrir:** Abre un proyecto existente. La plataforma *Arduino IDE* también cuenta con unos proyectos ya realizados a modo de ejemplos para fortalecer el aprendizaje de este lenguaje de programación.
- **Salvar:** Guarda el proyecto actual con el mismo nombre con el que cuenta actualmente.
- **Monitor serie:** Esta opción solo funciona si se encuentra una placa conectada al ordenador. Abre una ventana emergente que nos muestra la información que se haya especificado en el código mediante una función para que se puedan observar los cambios en esa variable.

Figura 13. Atajos de la interfaz Arduino IDE

Seguidamente, se encuentra la pantalla principal de la aplicación. En la parte superior se sitúan las distintas pestañas con código que corresponden a un mismo proyecto. Esto es útil a la hora de realizar proyectos muy extensos ya que permite realizar subdivisiones en el código, de forma que resulta más cómodo si se desea transportar una parte del código a otros proyectos o simplemente si es preferible tenerlo bien distribuido para una mejor visualización de éste. A la derecha de la horizontal donde se sitúan las pestañas se encuentra otro menú desplegable relacionado con el manejo de las pestañas, ya sea crear nuevas, borrar algunas abiertas o seleccionar entre las que ya se encuentren abiertas para poder observar el código.

Figura 14. Menú de pestañas de la interfaz Arduino IDE

Una vez se crea un nuevo proyecto, en la pestaña que nos muestra la interfaz se observa unas líneas de código escritas a modo de guía.

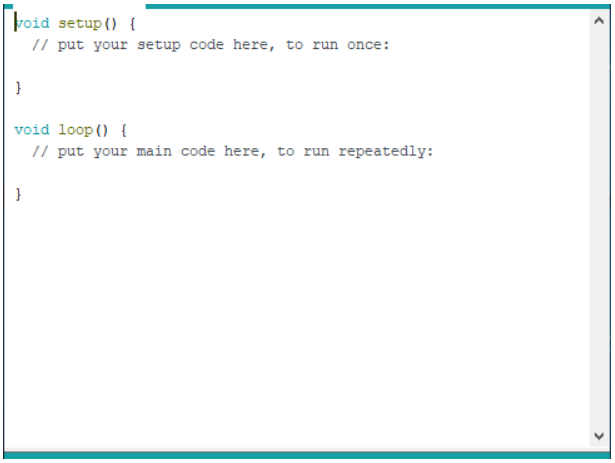

Figura 15. Sección de escritura de código de la interfaz Arduino IDE

Esta sección de la aplicación está dividida en dos funciones *void()*, que no devuelven ningún valor, distintas. Primeramente se encuentra la función *void setup()*, la cual se ejecuta solamente una vez al alimentarse la placa. Dentro de esta función se sitúa la parte del código encargada de realizar las configuraciones iniciales necesarias del proyecto. La segunda función *void loop()* es la encargada de ejecutar el código que se encuentre en su interior repetidamente de forma periódica, de modo que aquí se encontrará el código del proyecto que se encargue de realizar las tareas correspondientes a éste.

Como cualquier aplicación de escritura de código, éste se ejecuta de forma periódica de arriba abajo, de forma que si se desean introducir variables que sean necesarias para el proyecto además de las librearías correspondientes para acceder a funciones ya establecidas, es necesario que esa parte de código se encuentre por encima de las dos funciones *void()*, de forma que primero se creen estas variables y después se ejecute el resto del código.

Por último, en la parte inferior de la aplicación se encuentra la ventana de errores y advertencias. Ésta permite visualizar los errores en la compilación para proceder a su corrección o puede avisar de alguna advertencia con respecto a éste. También indica si se ha producido algún error durante la carga del código en la placa.

Figura 16. Sección de errores y advertencias de la interfaz Arduino IDE

3.2.2. ANDROID STUDIO

Para la programación de la aplicación móvil se ha utilizado el programa *Android Studio* versión 3.4.1. Éste, al igual que el *software* usado para la programación de la placa electrónica, también es un programa de código abierto totalmente gratuito basado en *Java*, por lo que el lenguaje de programación que utiliza es una variante de éste . Esto permite a cualquier usuario aprender acerca de este lenguaje y a crear sus propias aplicaciones móviles, lo que puede resultar útil en su vida laboral.

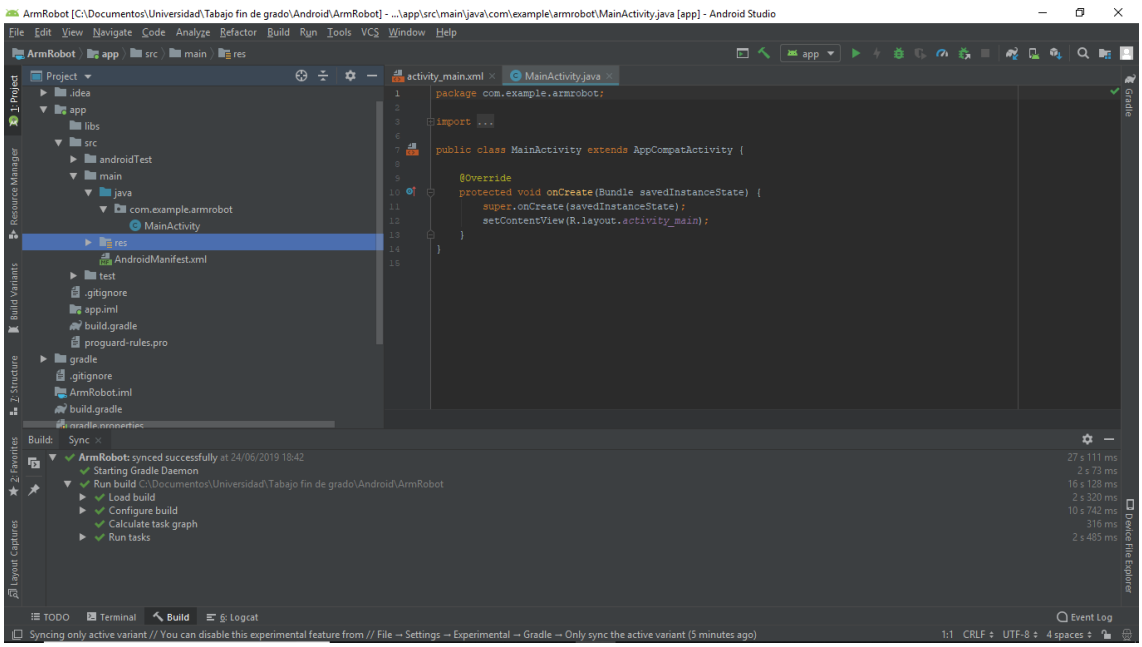

Figura 17. Interfaz Android Studio

La interfaz que presenta esta aplicación se divide en distintas secciones. Como todas las aplicaciones de escritorio, en la parte superior disponemos del nombre del proyecto, en este caso también aparece la ruta en la que se encuentra (factor muy a tener en cuenta a la hora de utilizar este lenguaje de programación), y menús desplegables que permiten acceder a todas las funciones que proporciona este programa.

Figura 18. Menús desplegables de la interfaz Android Studio

Sobre el lado izquierdo, la aplicación muestra todas las carpetas de las que está formada el proyecto, de forma que navegando entre ellas se accede a los distintos archivos que componen el proyecto y que son necesarios para realizar una aplicación. En todo momento la aplicación muestra en qué directorio se encentra el usuario mostrando en la parte superior las carpetas en las que se encuentra. En la parte izquierda se pueden seleccionar diversos menús que cambian esta ventana por otras para acceder a otras configuraciones más avanzadas.

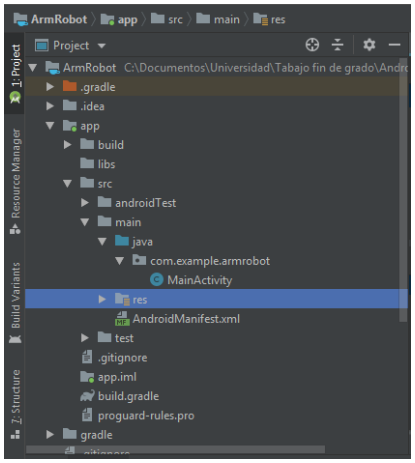

Figura 19. Sección con los directorios de la aplicación de la interfaz Android Studio

Sobre el lado derecho, el programa muestra los distintos archivos que se encuentren abiertos, ya sean los correspondientes al código cuya función es ejecutar el programa una vez se instale en el dispositivo móvil (archivos .java) o los correspondientes a la parte visual de la aplicación (archivos .xml). En este segundo caso, la ventana cambiará para poder tener una vista previa de la aplicación, además de que aparecerá a la derecha otro menú que permite la modificación de las características de los objetos que se vayan introduciendo en la aplicación. Cabe comentar que para os archivos .xml esta plataforma permite realizar un cambio a la hora de programarlos, de forma que es posible tanto programarlos mediante código o diseñarlo con un formato más visual. Se puede cambiar entre estas dos vistas gracias a una pequeña pestaña situada en la parte inferior izquierda de la sección. Además, en la parte superior se encuentran una serie de atajos para algunas opciones más utilizadas en los proyectos.

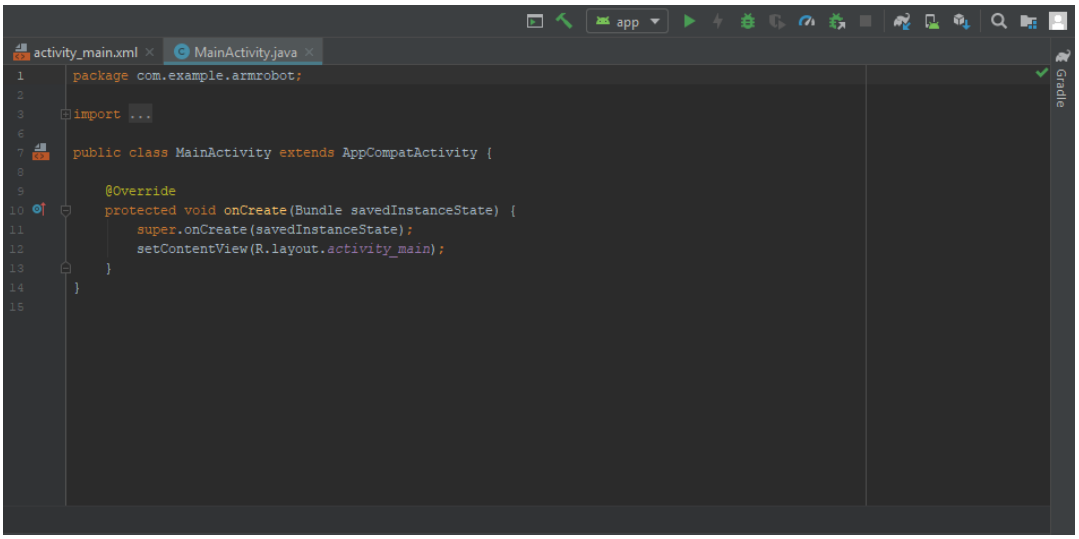

Figura 20. Sección de escritura de código de la aplicación de la interfaz Android Studio

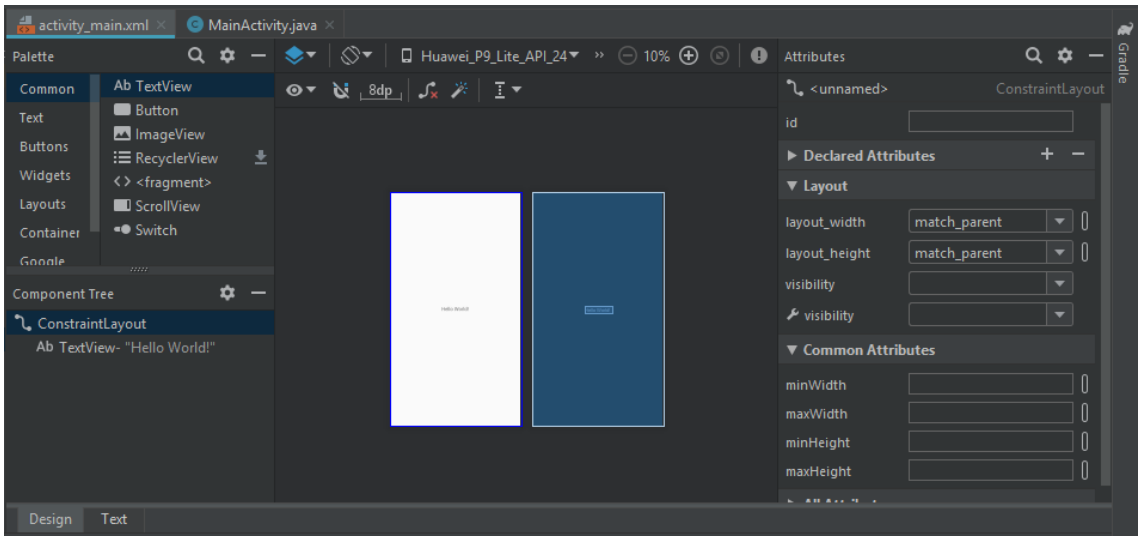

Figura 21. Sección de diseño de la aplicación de la interfaz Android Studio

Por último, en la parte inferior de la aplicación se encuentra la ventana de errores y advertencias, de modo que, si existe algún error durante la compilación del programa o durante la carga de éste en el dispositivo, se mostrará en esta ventana para proceder a su corrección.

Figura 22. Sección de errores y advertencias de la interfaz Android Studio

4. Descripción detallada de la solución

4.1. MONTAJE

Una vez explicados todos los elementos necesarios para la realización del proyecto, tanto el *hardware* utilizado como el *software*, el siguiente paso es la descripción de los pasos realizados para el funcionamiento de la aplicación, siendo estos el montaje del brazo robótico, la distribución de las conexiones y la programación de las funciones que realiza el brazo y de la aplicación del dispositivo móvil.

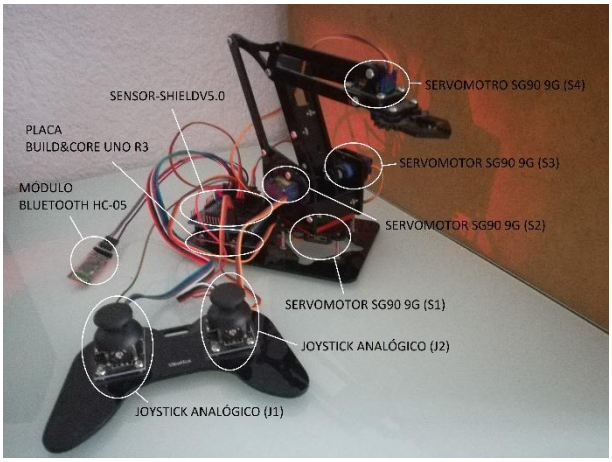

Figura 23. Montaje final del kit Arm Robot

El primer paso ha sido realizar el montaje del brazo robótico educativo de *Ebotics* siguiendo las instrucciones de montaje explicadas detalladamente en el manual que viene incluido en el kit. También se ha incorporado una pieza adicional para intentar disminuir un pequeño error en el diseño de la estructura del brazo, concretamente para tener una base más sólida y ayudar al servomotor encargado de la rotación del brazo (S1) a no tener que soportar enteramente la estructura de éste.

A la vez que se procedía al montaje del brazo, también se ha planteado la conexión de los distintos elementos con la placa *Sensor-ShieldV5.0*, factor a tener en cuenta a la hora de realizar la programación de éste en la aplicación *Arduino IDE.*

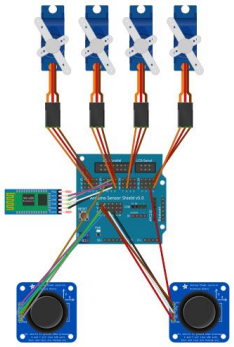

Figura 24. Conexionado de los elementos del kit de robótica Arm Robot

El direccionamiento de los distintos elementos ha sido el siguiente:

MÓDULO	PATILLA	TIPO DE PIN	NÚMERO DE PIN
SERVOMOTOR S1	-	DIGITAL	8
SERVOMOTOR S2	-	DIGITAL	2
SERVOMOTOR S3	-	DIGITAL	13
SERVOMOTOR S4	-	DIGITAL	6
BLUETOOTH HC-05	RXD	DIGITAL	10
	TXD	DIGITAL	9
JOYSTICK J1	VRX	ANALÓGICO	A1
	VRY	ANALÓGICO	A0
	SW	DIGITAL	11
JOYSTICK J2	VRX	ANALÓGICO	A3
	VRY	ANALÓGICO	A2
	SW	DIGITAL	3

Figura 25. Direccionamiento de los elementos conectados a la placa

Las patillas correspondientes a la alimentación y la masa de los distintos módulos se pueden conectar a cualquier pin que sirvan para tal efecto. Normalmente se sitúan pegados a los pines destinados a las señales de forma que facilita el cableado con la placa, como ha sido este caso en el que estas patillas de alimentación y masa se han direccionado lo más cercano posible a sus correspondientes señales.

4.2. PROGRAMACIÓN DE LA APLICACIÓN MÓVIL BASADA EN ANDROID

Para entender mejor la conexión entre las dos programaciones realizadas en la placa electrónica y en la aplicación móvil respectivamente, se verá primero a desarrollada en *Android Studio*,

debido a que es la encargada de enviar las órdenes a la placa, y ésta a los actuadores (los servomotores) del brazo robótico.

4.2.1. ACTIVITY

Primero es necesario conocer el concepto básico de *Activity* (Actividad), debido a que las aplicaciones basadas en *Android* funcionan mediante éstas.

Debido a que el lenguaje de programación de *Android* está basado en el de *Java* (programación orientada a objetos), este primero hereda los principios básicos del segundo, de forma que las *Activities* se definen como clases. Esto les permite asignarle atributos y funciones propias de esa clase para poder crear diferentes objetos, que serán las distintas *Activities*, cada una con sus características propias, a partir de una misma clase, la clase *Activity*.

De esta forma, una aplicación puede estar formada por varias *Activities*, siendo estas las distintas ventanas de las que va a estar formada. Éstas se van almacenan en una pila conforme se van ejecutando, de forma que la que esté en primer lugar será la ejecutada por la aplicación. Debido a esto, cada *Activity* de la aplicación se puede encontrar en un estado diferente, siendo estos:

- **Ejecución (*Running*):** Si se encuentra encima de la pila es la mostrada por la aplicación.
- **Pausa (*Paused*):** Se da cuando otra *Activity* se encuentra delante.
- **Parada (*Stopped*):** Este estado se da si otra *Activity* la tapa por completo. Aún conserva información, aunque no sea mostrada por la aplicación.
- **Finalizada (*Finished*):** La *Activity* será finalizada cuando, estando en pausa o parada, sea eliminada por el sistema cerrando el proceso.

Toda *Activity* está formada por dos archivos, uno .java y otro .xml. En el primero es donde se va a encontrar toda la programación de la aplicación: la creación de la *Activity*, las diferentes funciones de las que va a disponer, etc. Por otra parte, el segundo archivo es el correspondiente a la parte visual de esa *Activity*: la localización de los botones, los textos y demás elementos visuales. Esto permite que el archivo .java principal, el que se ejecuta al iniciar la aplicación, llame mediante unas funciones a archivos .xml para que los muestre cuando sea oportuno.

4.2.2. CREAR NUEVO PROYECTO

Al crear un nuevo proyecto, la plataforma *Android Studio* ofrece una gran cantidad de opciones avanzadas para personalizar la aplicación a desarrollar como mejor convenga, desde en qué tipo de dispositivos se va a utilizar o qué diseño visual predeterminado va a tener la *Activity* principal hasta en qué lenguaje de programación se va a desarrollar la aplicación (*Java* o *Kotlin*) y qué nivel *API* va a requerir la aplicación, en el caso de este proyecto es la *API* 24 correspondiente a la versión de *Android* 7.0 o *Nougat* debido a que es la versión que usa el dispositivo móvil en el que se va a probar la aplicación. Esto es importante para saber en qué porcentaje de dispositivos *Android* se podrá ejecutar la aplicación.

Otro factor importante a la hora de crear un nuevo proyecto es el *package* (paquete), en el cual se van a encontrar los elementos necesarios para ejecutar la aplicación. *Android* necesita de estos paquetes debido a que sirven como librerías, de forma que se pueden incorporar al proyecto distintos paquetes para luego llamarlos y buscar alguna función que se desee utilizar

que ya esté desarrollada en el nuevo paquete incorporado. En el caso de este proyecto el nombre del paquete en el que se guardará será *com.zoruda.proyectotfg*.

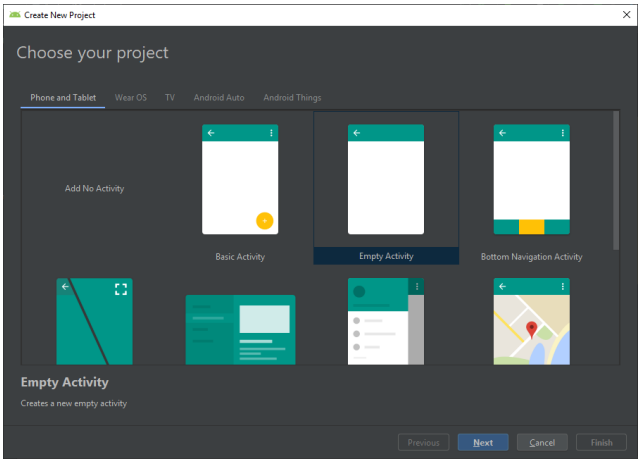

Figura 26. Selección Activity principal de nuevo proyecto

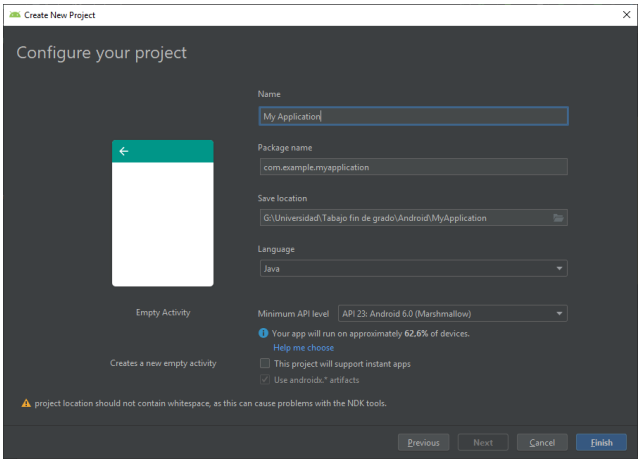

Figura 27. Ajuste creación de nuevo proyecto

Una vez creado el nuevo proyecto, el siguiente paso ya es comenzar con el desarrollo de la aplicación. Para ello es necesaria la modificación de tres archivos distintos creados automáticamente por el programa: *AndroidManifest.xml*, *activity_main.xml* y *MainActivity.java*.

4.2.3. ANDROIDMANIFEST.XML

Este archivo es el encargado de activar los permisos que requiera usar la aplicación para su funcionamiento además de los requisitos *Hardware* de la aplicación como la declaración de las *activities* que se vayan a diseñar. También incorpora algunas características propias de la aplicación, como el nombre que se mostrará al abrirla, el icono que la identificará en el dispositivo móvil o si por defecto se mostrará horizontal o verticalmente.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.zoruda.proyectotfg">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Arm Robot"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
            android:screenOrientation="landscape">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Figura 28. Archivo AndroidManifest.xml

En el caso de este proyecto, el archivo *AndroidManifest.xml* se ha modificado de forma que tenga los permisos necesarios para permitir la conexión *buetooth*, que la aplicación muestre el nombre *Arm Robot* al ejecutarse y que la *activity* principal esté por defecto en horizontal.

4.2.4. ACTIVITY_MAIN.XML

El archivo *activity_main.xml* es el correspondiente a la parte visual de la aplicación desarrollada (botones, textos, etc). Cada elemento visual que se puede introducir en este archivo se denomina vista.

El archivo dispone de dos formas de diseñarlo: mediante código o, de forma más visual, arrastrando objetos a un cuadro en blanco de forma que se puede visualizar el resultado más fácilmente.

La ventaja de diseñarlo de esta segunda forma es que resulta más cómodo y más visual a la hora de situar los distintos elementos y a hora de ajustar las distintas opciones de las que dispone cada objeto, así como asociarlos a las funciones que se hayan desarrollado en el código de la aplicación. Cualquier modificación en alguna de las dos pestañas de las que dispone este archivo modifica de forma automática la otra pestaña, de forma que siempre son equivalentes.

Para el diseño de la aplicación del proyecto, se ha optado por una ventana principal simple, que solamente incluye botones que se activarán y se desactivarán en función de en qué modo se encuentre la aplicación.

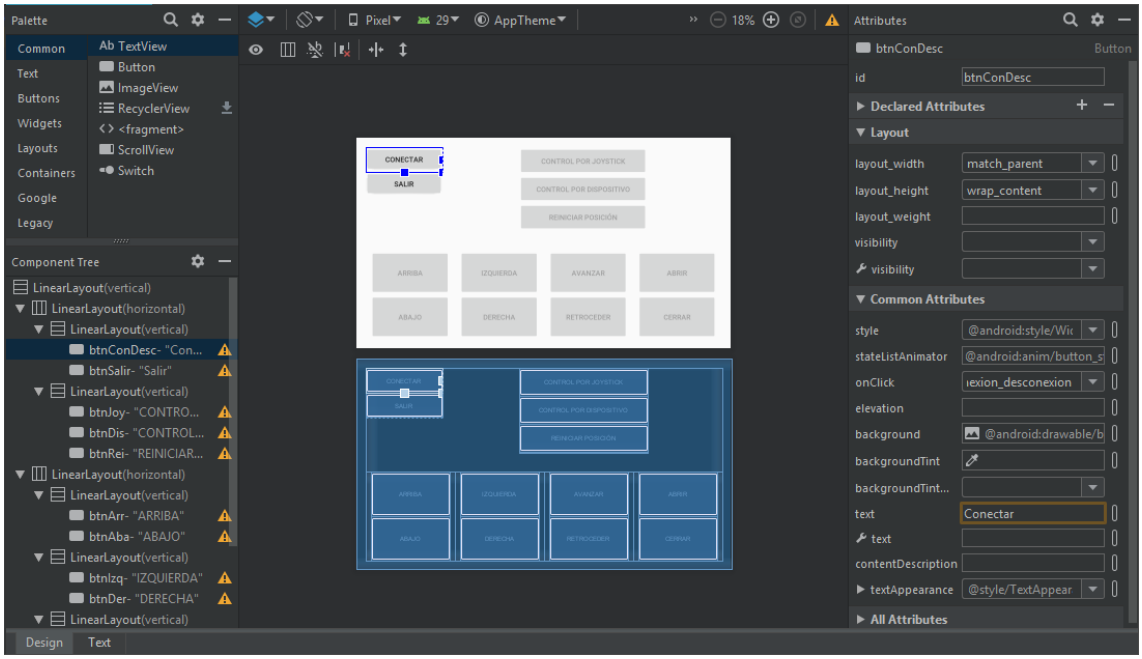

Figura 29. Diseño del archivo activity_main.xml

El diseño planteado implementa las vistas:

- **Botón "CONTECTAR":** Realiza la conexión *bluetooth* con el dispositivo móvil.
- **Botón "CONTROL POR *JOYSTICK*":** Habilita el control del brazo con los *joysticks* analógicos conectados a la periferia de la placa.
- **Botón "CONTROL POR DISPOSITIVO":** Habilita el control del bazo con el dispositivo móvil, de forma que se pasa a un control de forma remota.
- **Botón "REINICIAR POSICIÓN":** Ordena al brazo a regresar a su posición inicial.
- **Botones "ARRIBA", "ABAJO", "IZQUIERDA", "DERECHA", "AVANZAR", "RETROCEDER", "ABRIR" y "CERRAR":** Mueven el brazo estando activado el control por dispositivo.
- **Botón "SALIR":** Cierra la aplicación.

4.2.5. MAINACTIVITY.JAVA

Este es el archivo principal que se encarga del correcto funcionamiento de la aplicación. Aquí se desarrollan las funciones necesarias para realizar las distintas tareas de las que dispone la aplicación, se asignan los distintos objetos que se han utilizado en el archivo *activity_main.xml*, así como las variables que de las que se ha hecho uso en las diferentes funciones desarrolladas.

Para el desarrollo del código, primero incluir tanto el paquete en dónde se encuentran los archivos necesarios para su ejecución como las distintas librerías, que ya incluye el programa *Android Studio*, para su uso en caso de querer implementar alguna función que haga uso de una de éstas.

```
package com.zoruda.proyectotfg;

import java.io.IOException;
import java.io.OutputStream;
import java.lang.String;
import java.util.Set;
import java.util.UUID;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
```

Figura 30. Paquete y librerías incluidas en la programación

Android Studio facilita el añadido de estas librerías gracias a que, si se llama a una función que no tenga la librería incluida en el programa, la importa automáticamente.

Una vez declarados todas las librerías, a continuación se crea la clase *MainActivity* que extiende la clase *AppCompatActivity* de forma que se puede hacer uso de sus funciones. Seguidamente, y ya dentro de la clase definida, se inicializan todas las variables que se van a utilizar en las diferentes funciones programadas.

```
public class MainActivity extends AppCompatActivity {


    /*-----ACTIVAR-----
    -------OPCIONES--*/
    BluetoothAdapter mAdapter;
    BluetoothSocket mSocket;
    BluetoothDevice mDevice;

    OutputStream mOutpoutStream;


    /*-----INICIALIZACION-----
    -------VARIABLES--------*/
    Button mBJoy = null;
    Button mBDis = null;
    Button mBArr = null;
    Button mBAba = null;
    Button mBDer = null;
    Button mBIzq = null;
    Button mBAva = null;
    Button mBRet = null;
    Button mBCer = null;
    Button mBAbr = null;
    Button mBRei = null;

    String address = null;

    UUID uuid = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    boolean a = true;
    boolean b = true;
    boolean c = true;
```

Figura 31. Creación clase MainActivity e inicialización de variables

Para el proyecto se han activado las opciones necesarias para permitir la conexión *bluetooth*, se han inicializado los objetos de la clase *Button* de los que se ha hecho uso, además de otros objetos de otras clases, como un *String*, un *UUID* (Identificador Único Universal) y varios booleanos necesarios en algunas funciones.

A continuación, se crea la *Activty* principal y llama al archivo *activity_main.xml*, de forma que lo muestra al ejecutarse. Dentro de esta *Activity* se asigna cada ID de los botones introducidos en el archivo *activity_main.xml* a cada objeto botón creado en *MainActivity.java*, de forma que el

programa relaciona ambos archivos del proyecto. Esto es posible gracias a la función *findViewById,* la cual busca la ID en el directorio donde se almacenan todas (en R.id.*).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);


    /*-----ASIGNACIÓN-----
    -------DE BOTONES---*/
    mBJoy = findViewById(R.id.btnJoy);
    mBDis = findViewById(R.id.btnDis);
    mBArr = findViewById(R.id.btnArr);
    mBAba = findViewById(R.id.btnAba);
    mBDer = findViewById(R.id.btnDer);
    mBIzq = findViewById(R.id.btnIzq);
    mBAva = findViewById(R.id.btnAva);
    mBRet = findViewById(R.id.btnRet);
    mBCer = findViewById(R.id.btnCer);
    mBAbr = findViewById(R.id.btnAbr);
    mBRei = findViewById(R.id.btnRei);
```

Figura 32. Creación de la Activity principal y asignación de botones

También se encuentran las funciones que dan la orden a la placa electrónica de *Arduino* de realizar los correspondientes movimientos del brazo robótico. La función utilizada para tal efecto es una propia de la clase botón, *setOnTouchListener*. Esta función deja al botón a la espera de que haya alguna interacción con él. Para el caso del proyecto, se ha modificado de forma que una vez detecta dicha interacción, evalúa si corresponde a la pulsación o a la liberación del botón. Según el caso, enviará una variable de tipo *string* (una cadena de caracteres) vía *bluetooth* gracias a la función *sendBT(i)*, desarrollada más abajo en el código. Según la letra enviada a la placa electrónica, ésta realiza un movimiento u otro. Es necesario que estas funciones se encuentren dentro de esta *Activity* debido a que se está sobrescribiendo continuamente, de forma que se mantiene el botón pulsado, se envía continuamente el *String* correspondiente, necesario para el correcto movimiento del brazo robótico.

```
/*-----SERVOMOTOR 1-----*/

/*-----MOVIMIENTO DERECHA-----*/
mBDer.setOnTouchListener((v, event) -> {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            sendBT( B "A");
        }
        if (event.getAction() == MotionEvent.ACTION_UP) {
            sendBT( B "C");
        }
        return true;
});

/*-----MOVIMIENTO IZQUIERDA-----*/
mBIzq.setOnTouchListener((v, event) -> {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            sendBT( B "B");
        }
        if (event.getAction() == MotionEvent.ACTION_UP) {
            sendBT( B "C");
        }
        return true;
});
```

Figura 33. Ejemplo función para el envío de la variable String a la placa electrónica

Para cada uno de los cuatro servomotores de los que dispone el brazo robótico se ha seguido una estructura similar en la programación a la que se puede observar en la *Figura 32* debido a que siguen el mismo procedimiento, dependiendo del botón que se pulse o se suelte se enviará un *String* distinto.

Las funciones restantes implementadas se encuentran fuera de la *Activity* debido a que no es necesario que estén en ejecución constante, ya que solo se van a llamar una vez al pulsar los botones correspondientes.

La función encargada de realizar la conexión *bluetooth* es la llamada *startBT()*. El procedimiento para conseguir dicha conexión es el siguiente: primero define el objeto *Adapter*, el cual es el adaptador *bluetooth* del que dispone el dispositivo móvil.

A continuación, evalúa si ese dispositivo no dispone de ese adaptador, mostrando el mensaje "Este dispositivo no dispone de conexión Bluetooth" en caso afirmativo, de forma que no se puede realizar la conexión *bluetooth*.

Seguidamente comprueba si, teniendo este adaptador, se encuentra deshabilitado, en cuyo caso muestra un mensaje de que el modo *bluetooth* del dispositivo no está activo y se pregunta al usuario si desea habilitarlo. La función que muestra este mensaje ya viene incluida en una de las librearías de *Android Studio*.

Después busca los dispositivos *bluetooth* que se encuentren al alcance, de forma que si encuentra alguno, se asigne automáticamente a uno que tenga la dirección establecida en el programa: "98:D3:32:30:F7:5F" (ver *4.3.1 Configuración módulo bluetooth HC-05*).

La última parte de la función se encarga de crear un objeto *Device,* siendo este el elemento que ha encontrado el adaptador con la dirección asignada, para luego crear un objeto *Socket* (un canal para transmitir la información) con ese *Device* asignándole al *Socket* la *UUID* asignada al principio del programa. Después, se conecta el *Socket* y se crea el objeto *OutputStream*, el cual permite el envío de información por el *Socket* al dispositivo *bluetooth* conectado a la placa electrónica.

Como añadido extra, se habilitan los botones correspondientes a los modos de control del brazo robótico, así como se muestra un mensaje si la conexión ha sido completada o si ha habido algún fallo en ésta y no se ha podido completar, en cuyo caso no se habilitarán los botones.

La variable booleana “c” simplemente define el texto que se muestra en el botón que ejecuta esta función, si “c” es cierta (*true*) el texto del botón muestra “DESCONECTAR” y si es falso (*false*) muestra “CONECTAR”.

```
203     public void startBT() throws IOException {
204
205         mAdapter = BluetoothAdapter.getDefaultAdapter();
206
207         if (mAdapter == null) {
208             showMessage( msg: "Este dispositivo no dispone de conexión Bluetooth", Toast.LENGTH_LONG
209             return;
210         }
211
212         if (!mAdapter.isEnabled()) {
213             Intent enableBluetooth = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
214             startActivityForResult(enableBluetooth,  requestCode: 1);
215         }
216
217         Set<BluetoothDevice> pairedDevices = mAdapter.getBondedDevices();
218
219         if (pairedDevices.size() > 0) {
220             for (BluetoothDevice device : pairedDevices) address = "98:D3:32:30:F7:5F";
221         }
222
223         try {
224             mDevice = mAdapter.getRemoteDevice(address);
225             mSocket = mDevice.createInsecureRfcommSocketToServiceRecord(uuid);
226             mSocket.connect();
227             mOutpoutStream = mSocket.getOutputStream();
228             mBDis.setEnabled(true);
229             mBJoy.setEnabled(true);
230             showMessage( msg: "Bluetooth conectado", Toast.LENGTH_SHORT);
231             c = true;
232         } catch (IOException e) {
233             showMessage( msg: "Fallo en el intento de conexión", Toast.LENGTH_SHORT);
234             c = false;
235         }
236     }
```

Figura 34. Función startBT()

Por otra parte, la función que se de finalizar la conexión *bluetooth*, *endBT()*, es mucho más sencilla. Simplemente llama a la función *close()* tanto el objeto *OutputStream* como el *Socket* creados en la función anterior, de forma que los cierra. También cambia el valor de la variable "c" a falso para que el texto del botón cambie a "CONECTAR" y deshabilita todos los botones excepto los correspondientes a la conexión *bluetooth* y al cierre de la aplicación.

```
242     public void endBT() throws IOException {
243         mOutpoutStream.close();
244         mSocket.close();
245         c = false;
246
247         mBDis.setEnabled(false);
248         mBJoy.setEnabled(false);
249         mBArr.setEnabled(false);
250         mBAba.setEnabled(false);
251         mBDer.setEnabled(false);
252         mBIzq.setEnabled(false);
253         mBAva.setEnabled(false);
254         mBRet.setEnabled(false);
255         mBAbr.setEnabled(false);
256         mBCer.setEnabled(false);
257         mBRei.setEnabled(false);
258     }
```

Figura 35. Función endBT()

Para ejecutar estas dos funciones anteriores, se ha programado otra función, *conexion_desconexion(v)*, que será llamada al pulsar el botón "CONECTAR".

Esta nueva función se encarga de comprobar dos casos: si existe *Socket* y está conectado o si no existe, por lo que no puede estar conectado. Para el primer caso ejecutará la función *endBT()*, explicada anteriormente.

En cambio, para el segundo caso ejecuta la función contraria, *startBT()*. También muestra mensajes en caso de haber algún fallo durante la ejecución de la función, además de cambiar el texto de botón "CONECTAR" en función de en qué estado se encuentre la aplicación, si está conectada a la placa o no.

Figura 36. Función conexion_desconexion(v)

Para poder mostrar los mensajes que indican si ha sido posible realizar la conexión o ha habido algún fallo que lo ha impedido, se ha desarrollado una función denominada *showMessage(msg, time)*. Ésta se encarga de mostrar estos mensajes emergentes, llamados *Toast*, de forma que cada vez que se quiera mostrar un mensaje emergente, se llama a esta función asignándole el texto a mostrar y el tiempo que se ve en la pantalla.

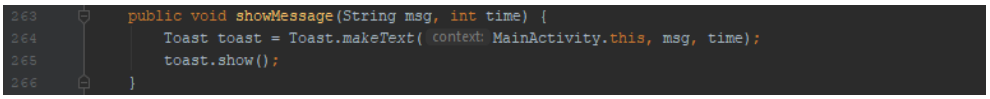

Figura 37. Función showMessage(msg, time)

La función que envía los distintos caracteres, *sendBT(i)*, simplemente evalúa si se ha definido un *Socket* y, en caso afirmativo, escribe en el *OutputStream* un variable de tipo *String* (conjunto de caracteres).

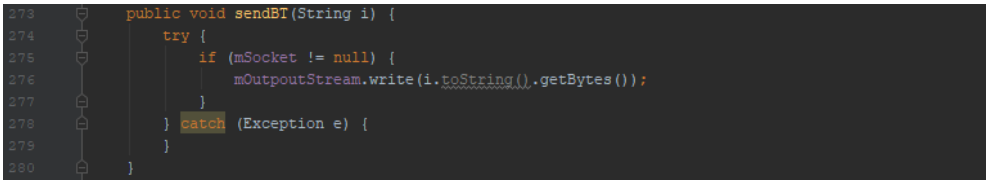

Figura 38. Función sendBT(i)

La función para controlar el brazo robótico mediante el dispositivo móvil, *modo_dispositivo(v)*, simplemente habilita y deshabilita los botones correspondientes dependiendo de si ya se encontraba en este modo o no.

```
286     public void modo_dispositivo(View v) {
287         if (a) {
288             mBDis.setText("DESACTIVAR MODO");
289             mBArr.setEnabled(a);
290             mBAba.setEnabled(a);
291             mBDer.setEnabled(a);
292             mBIzq.setEnabled(a);
293             mBAva.setEnabled(a);
294             mBRet.setEnabled(a);
295             mBCer.setEnabled(a);
296             mBAbr.setEnabled(a);
297             mBRei.setEnabled(a);
298             mBJoy.setEnabled(!a);
299         } else {
300             mBDis.setText("CONTROL POR DISPOSITIVO");
301             mBArr.setEnabled(a);
302             mBAba.setEnabled(a);
303             mBDer.setEnabled(a);
304             mBIzq.setEnabled(a);
305             mBAva.setEnabled(a);
306             mBRet.setEnabled(a);
307             mBCer.setEnabled(a);
308             mBAbr.setEnabled(a);
309             mBRei.setEnabled(a);
310             mBJoy.setEnabled(!a);
311         }
312         a = !a;
313     }
```

Figura 39. Función modo_dispositivo(v)

En cambio, la función para activar el control del brazo mediante los *joysticks* analógicos, *modo_joystick(v)*, realiza una tarea similar a las funciones encargadas del movimiento del brazo, puesto que envían una variable de tipo *String* a la placa dependiendo de si está habilitado o no, además de modificar el texto del propio botón.

```
319     public void modo_joystick(View v) {
320         if (b) {
321             mBJoy.setText("DESACTIVAR MODO");
322             mBDis.setEnabled(!b);
323             sendBT( R "M");
324         } else {
325             mBJoy.setText("CONTROL POR JOYSTICK");
326             mBDis.setEnabled(!b);
327             sendBT( R "N");
328         }
329         b = !b;
330     }
```

Figura 40. Función modo_joystick(v)

De la misma forma, la función “reinicio”, envía otro *String* distinto para que la placa ejecute el proceso necesario para regresar el brazo robótico a la posición inicial.

```
334     public void reinicio(View v) {
335         sendBT( R "O");
336     }
```

Figura 41. Función reinicio(v)

Finalmente, la función *salir(v)* llama a *endBT()* para finalizar la conexión si estuviera establecida y también ejecuta la función *finish()* que cierra la aplicación.

```
342     public void salir(View v) throws IOException {
343         endBT();
344         finish();
345     }
```

Figura 42. Función salir(v)

Los distintos *Strings* que se envían a la placa electrónica dependen de la función que se ejecute y que se quiera que se realice en la placa, siguiendo el siguiente esquema:

***STRING* ENVIADO**	**ACTIVACIÓN**	**FUNCIÓN**
A	Mantener botón "DERECHA"	Girar el brazo hacia la derecha
B	Mantener botón "IZQUIERDA"	Girar el brazo hacia la izquierda
C	Soltar botón "DERECHA" o "IZQUIERDA"	Parar el giro
D	Mantener botón "AVANZAR"	Inclinar el brazo hacia delante
E	Mantener botón "RETROCEDER"	Inclinar el brazo hacia detrás
F	Soltar botón "AVANZAR" o "RETROCEDER"	Parar la inclinación
G	Mantener botón "ARRIBA"	Mover el brazo hacia arriba
H	Mantener botón "ABAJO"	Mover el brazo hacia abajo
I	Soltar botón "ARRIBA" o "ABAJO"	Parar el movimiento
J	Mantener botón "ABRIR"	Abrir la pinza
K	Mantener botón "CERRAR"	Cerrar la pinza
L	Soltar botón "ABRIR" o "CERRAR"	Parar el movimiento de la pinza
M	Pulsar botón "CONTROL POR JOYSTICK"	Si el modo se encuentra desactivado, lo activa
N	Pulsar botón "CONTROL POR JOYSTICK"	Si el modo se encuentra activado, lo desactiva
O	Pulsar botón "REINICIAR POSICIÓN"	Reinicia la posición del brazo

Figura 43. Funciones de los String enviados

4.2.6. INSTALACIÓN EN EL DISPOSITIVO MÓVIL

Por último, para poder instalar la aplicación desarrollada en el dispositivo móvil es necesario activar en los ajustes avanzados de éste la opción "Depuración USB", la cual permite la instalación de aplicaciones desde *Android Studio*, así como intercambiar datos entre el ordenador y el dispositivo móvil. Cabe destacar que también es necesaria la instalación del *Android SDK*, un conjunto de herramientas que permiten el desarrollo de aplicaciones para un sistema *Android*, para poder realizar la instalación de la aplicación.

Esta opción viene oculta por defecto en los dispositivos móviles en un menú también oculto llamado "Opciones del desarrollador". Para activar este menú, simplemente hay que ir a los ajustes del dispositivo, ir a "Acerca del teléfono" y pulsar siete veces sobre el "Número de compilación".

Al realizar esto, en los ajustes aparecerá el menú “Opciones del desarrollador”, en el cual se encontrará el ajuste necesario para poder instalar la aplicación, la “Depuración USB”.

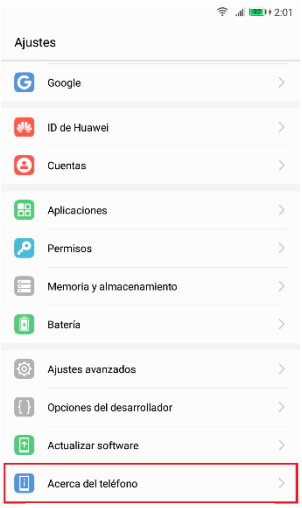

Figura 44. Menú "Ajustes"

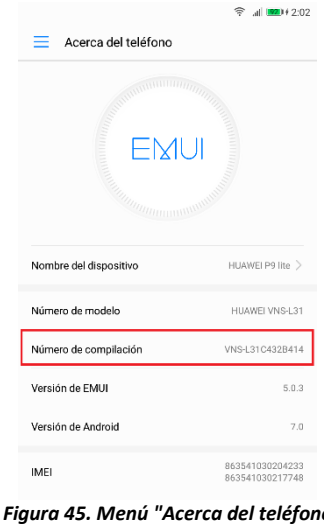

Figura 45. Menú "Acerca del teléfono"

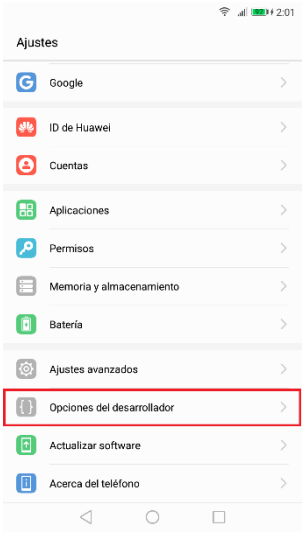

Figura 46. Opciones del desarrollador

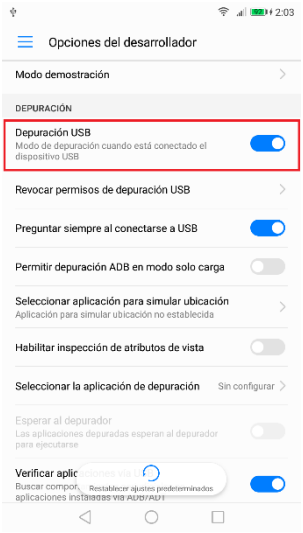

Figura 47. Activación "Depuración USB"

4.3. PROGRAMACIÓN DE LA PLACA ELECTRÓNICA BASADA EN ARDUINO

El programa implementado en la placa electrónica basada en *Arduino*, a diferencia de la programación en *Android Studio*, consta solo de un archivo con extensión *.ino* al que se ha denominado *ArmRobot.ino*. Pero antes de comenzar con la programación, es necesario realizar la configuración del módulo *bluetooth HC-05*.

4.3.1. CONFIGURACIÓN MÓDULO BLUETOOTH HC-05

Para poder configurar el módulo *bluetooth HC-05* de *Arduino*, es necesario hacerlo mediante los comandos *AT*. Estos comandos son una serie de órdenes que se le dan al módulo mediante un

monitor serie y que permiten modificar parámetros propios de éste, como el papel que desempeña en la conexión (esclavo o maestro), el nombre del dispositivo, la velocidad de comunicación, así como obtener datos de interés, como la dirección del módulo.

DESCRIPCIÓN	COMANDO AT	FUNCIÓN	RESPUESTA
Test de comunicación	AT	Comprobación si el módulo responde a los comandos	OK
Cambio de nombre	AT+NAME=<Nombre>	Cambio de nombre del módulo. Por defecto se llama *HC-05*	OK
Configuración de la velocidad de comunicación	AT+UART=<Baud>, <StopBit>, <Parity>	Cambia la velocidad de comunicación del módulo, el bit de parada y la paridad. Por defecto es de 9600 baudios	OK
Preguntar por la velocidad actual	AT+UART?	Pregunta por la velocidad de comunicación, el bit de parada y la paridad con los que está configurado actualmente el módulo	+UART:<Baud>, <StopBit>, <Parity> OK
Configuración del papel	AT+ROLE=<Role>	Cambia el papel que desempeña el módulo en la comunicación. Por defecto viene como esclavo	OK
Preguntar por el papel actual	AT+ROLE?	Pregunta por el papel que desempeña actualmente en la comunicación	+ROLE:<Role> OK
Preguntar por la dirección del módulo	AT+ADDR?	Obtiene la dirección del módulo	+ADDR:<dirección> OK
Reset del módulo	AT+RESET	Resetea el módulo y sale del modo *AT*	OK
Restablecer valores predeterminados	AT+ORGL	Restablece todos los valores del módulo a los establecidos por defecto de fábrica	OK

Figura 48. Algunos de los comandos AT más importantes

Para poder introducir estos comandos, primero es necesario activar el modo correspondiente, ya que el módulo *bluetooth HC-05* dispone de varios modos de funcionamiento, siendo estos:

- **Desconectado:** El módulo entra en este estado tan pronto como se alimenta, y permanece en este mientras no se haya establecido una conexión *bluetooth* con él. En este caso, el *LED* del que dispone parpadea rápidamente.
- **Conectado:** Se da cuando se establece una conexión con otro dispositivo *bluetooth*. El *LED* emite un doble parpadeo mientras el módulo se encuentra en este estado, además de transmitir por *bluetooth* al dispositivo todos los datos que se ingresan por el pin *Rx* y devolver por el pin *Tx* los que se reciben.
- **Modo *AT* 1:** Para entrar en este modo hay que pulsar el botón que viene incorporado en el módulo después de haberlo conectado y alimentado. A primera vista el *LED*

parpadea como si se encontrara en el modo desconectado, pero es capaz de recibir comando *AT* a la velocidad a la que esté configurado.

- **Modo *AT* 2:** También dispone de otra forma de llegar a introducir los comandos *AT* en el caso de que no se recuerde la velocidad con la que está configurado. Para entrar en este modo es necesario mantener pulsado el botón del módulo a la vez que se alimenta, de forma que se encienda con el botón presionado. También es posible acceder a este modo si al encenderlo se encuentra la patilla *KEY* alimentada. En este modo es necesario enviar los comandos *AT* a 38400 baudios para poder reconfigurar el módulo. Además, el *LED* parpadea lentamente.

Pero antes de cambiar el modo del dispositivo a uno de los modos necesarios para cambiar su configuración, es necesario diseñar un código que permita la lectura de los comandos *AT* por parte del módulo y la respuesta de éste.

```
#include <SoftwareSerial.h>

SoftwareSerial BT1(9, 10); //RX, TX

void setup()
   {
       Serial.begin(9600);
       Serial.println("Itroduce comandos AT:");
       BT1.begin(9600);
   }

void loop()
   {
       if (BT1.available())
           Serial.write(BT1.read());

       if (Serial.available())
           BT1.write(Serial.read());
}
```

Figura 49. Código de configuración del módulo bluetooth

Este código es el necesario para realizar tal tarea. Para esto incluye al principio la librería necesaria para poder acceder a las funciones para realizar intercambio de información con el dispositivo, asigna las patillas *Rx* y *Tx* del módulo a los pines 9 y 10 de la placa, configura la velocidad del monitor serie y del módulo, muestra un pequeño mensaje por el monitor, para corroborar que el programa se ha ejecutado correctamente, y se queda a la espera de que se escriba información por el monitor serie, de forma que si se escribe alguno de los comandos *AT*, el módulo responde, visualizándose en el monitor.

Una vez implementado este código y puesto el módulo en uno de los modos *AT*, ya se puede proceder a configurarlo mediante esos comandos.

Figura 50. Configuración del módulo bluetooth mediante comandos AT

Cabe destacar que para que tanto el módulo *bluetooth* como el monitor serie puedan comunicarse e intercambiar información, además de asignarle a los dos la misma velocidad de comunicación, es necesario seleccionar en el monitor serie "Ambos NL & CR", debido a que, de esta forma, introduce el comando *AT* al detectar el retorno de carro (pulsar botón *enter*) y realiza saltos de línea entre cada orden enviada.

La configuración establecida para el módulo *bluetooth HC-05* incorporado en el brazo robótico *Arm Robot* es la siguiente:

- **Velocidad de comunicación:** 9600 baudios. Simplemente utilizado, en el código implementado en la placa electrónica, para iniciar el módulo *bluetooth* con esa velocidad.
- **Papel a desempeñar en la comunicación:** Como ha devuelto *+ROLE:0* quiere decir que está configurado para desempeñar el papel de esclavo en la comunicación. En el caso de devolver *+ROLE:1*, el papel que tendría sería el de maestro. La diferencia entre ambos consiste en quién envía la información y quién la recibe, siendo el maestro capaz de realizar ambas funciones y el esclavo pudiendo solamente recibir información. Para el proyecto se ha optado por dejarlo como esclavo en la comunicación.
- **Dirección del módulo:** Este dato es el utilizado en el desarrollo de la aplicación móvil (ver *4.2.5 MainActivity.java*), debido a que es la dirección que busca la aplicación para poder realizar la conexión *bluetooth*. Está asignada la dirección por defecto, siendo esta "98:D3:32:30:F7:5F".

Esta es toda la configuración necesaria que se va a realizar sobre el módulo *bluetooth HC-05* de *Arduino*.

4.3.2. CALIBRACIÓN DE LOS JOYSTICKS ANALÓGICOS

También es necesario realizar una calibración de los *joysticks* analógicos para poder hacer una estimación de a partir de qué valor, comenzar a realizar el movimiento del servomotor. Para esto se ha diseñado un código que realiza esta lectura y la muestra por el monitor serie. Se ha ido cambiando el conexionado de cada eje de cada *joystick* para poder realizar la medición individualmente.

```
#include <Servo.h>

int valorX = 0; // VARIABLE DE LECTURA DEL EJE X
int pinJX = A0; // EJE X CONECTADO AL PIN ANALOGICO A0

void setup() {
Serial.begin (9600); // ACTIVAR COMUNICACIÓN POR PUERTO SERIE
}

void loop() {

valorX = analogRead (pinJX);
Serial.print ("X: ");
Serial.println (valorX); // MOSTRAR POR PANTALLA LOS VALORES DEL EJE X DE 0 A 1023
}
```

Figura 51. Calibración de los joysticks analógicos

Después de realizar las medidas, se ha optado por la realización de la siguiente calibración:

JOYSTICK	EJE	VALOR CALIBRACIÓN	SERVOMOTOR	MOVIMIENTO
1	X	<503	1	Giro derecha
		>543		Giro izquierda
	Y	<515	2	Inclinar hacia delante
		>555		Inclinar hacia atrás
2	X	<497	3	Mover hacia arriba
		>537		Mover hacia abajo
	Y	<400	4	Abrir pinza
		>600		Cerrar pinza

Figura 52. Calibración de los joysticks analógicos

4.3.3. ARMROBOT.INO

La programación de las placas electrónicas de *Arduino* utiliza un lenguaje parecido a las aplicaciones de *Android*, puesto que ambos son lenguajes orientados a objetos, aunque el de *Arduino* tiene las bases sentadas de C, pero a diferencia del programa desarrollado con *Android Studio*, en *Arduino* solamente es necesario modificar un archivo .ino, donde se encuentra todo el código del programa y el cual se carga a la placa electrónica.

Para el desarrollo del código, primero es necesario incluir las librerías necesarias, que ya vienen instaladas en el programa, para poder acceder a las distintas funciones que van a ser utilizadas

en el proyecto, siendo en este caso solamente la necesaria para poder declarar los servomotores y asignarles velocidades, *<Servo.h>*.

```
#include <Servo.h>
#include <SoftwareSerial.h>
```

Figura 53. Librerías incluidas en el proyecto

También se ha incluido la librería *<SoftwareSerial.h>* para poder declarar el dispositivo *bluetooth* que incorpora el proyecto.

A continuación se definen las distintas variables de las que se hace uso en el programa, siendo éstas desde las lecturas de los ejes de los *joysticks* analógicos, hasta algunas variables necesarias en la realización de algunas funciones, pasando por la configuración de los pines, la declaración de los cuatro servomotores, la creación de las variables que corresponden a los grados de giro de los servomotores y la declaración del módulo *bluetooth*.

El direccionamiento de los pines es el asignado según el conexionado realizado durante el montaje (ver *Figura 24. Conexionado de los elementos del kit de robótica Arm Robot*.

```
int valorX1 = 0;      // LECTURA DEL EJE X de J1
int valorY1 = 0;      // LECTURA DEL EJE Y de J1
int valorX2 = 0;      // LECTURA DEL EJE X de J2
int valorY2 = 0;      // LECTURA DEL EJE Y de J2
bool valorB1 = false; // LECTURA DEL PULSADOR de J1
bool valorB2 = false; // LECTURA DEL PULSADOR de J2
/*-------------------------------------------------------------------------------------------*/
int pinJX1 = A0; // PIN ANALOGICO A1 DEL EJE X de J1
int pinJY1 = A1; // PIN ANALOGICO A0 DEL EJE Y de J1
int pinJB1 = 11; // PIN DIGITAL 11 DEL PULSADOR de J1
int pinJX2 = A2; // PIN ANALOGICO A2 DEL EJE X de J2
int pinJY2 = A3; // PIN ANALOGICO A3 DEL EJE Y de J2
int pinJB2 = 3;  // PIN DIGITAL 3 DEL PULSADOR de J2
/*-------------------------------------------------------------------------------------------*/
Servo motor1; // DECLARAR S1 -- SERVO MOTOR 1
Servo motor2; // DECLARAR S2 -- SERVO MOTOR 2
Servo motor3; // DECLARAR S3 -- SERVO MOTOR 3
Servo motor4; // DECLARAR S4 -- SERVO MOTOR 4
/*-------------------------------------------------------------------------------------------*/
int grados1 = 160; // GRADOS DEL S1
int grados2 = 60;  // GRADOS DEL S2
int grados3 = 20;  // GRADOS DEL S3
int grados4 = 90;  // GRADOS DEL S4
/*-------------------------------------------------------------------------------------------*/
SoftwareSerial BT(9, 10); // DECLARAR DISPOSITIVO BLUETOOTH HC-05(RX, TX)
/*-------------------------------------------------------------------------------------------*/
int vel = 2;
bool estadoAnterior = false;
bool estadoActual;
int contador = 0;
char data;
```

Figura 52. Declaración de variables

Seguidamente se encuentra la parte del código que se ejecuta una sola vez al alimentar la placa. Este código es el que se localiza dentro de la función *void setup()* y contiene la configuración de la velocidad del módulo *bluetooth* y del puerto serie, para poder observar algunos datos, y de los pines digitales, además de incluir el direccionamiento de los pines de los servomotores, así como incluye la función de *Reinicio()*, de forma que al alimentar la placa, el brazo regrese a una posición predeterminada.

```
void setup() {
  /*-------------------------------------------------------------------------------------------*/
  BT.begin(9600); //INICIAR BLUETOOTH A 9600 BAUD
  Serial.begin(9600);
  /*-------------------------------------------------------------------------------------------*/
  motor1.attach (8);  // PIN DIGITAL PWM 8 DONDE ESTÁ CONECTADO EL S1
  motor2.attach (2);  // PIN DIGITAL PWM 1 DONDE ESTÁ CONECTADO EL S2
  motor3.attach (13); // PIN DIGITAL PWM 13 DONDE ESTÁ CONECTADO EL S3
  motor4.attach (6);  // PIN DIGITAL PWM 6 DONDE ESTÁ CONECTADO EL S4
  /*-------------------------------------------------------------------------------------------*/
  pinMode (pinJB1, INPUT_PULLUP);  // PIN DIGITAL 11 CON CONFIGURACION ENTRADA Y RESISTENCIA PULL UP
  pinMode (pinJB2, INPUT_PULLUP);  // PIN DIGITAL 3 CON CONFIGURACION ENTRADA Y RESISTENCIA PULL UP
  /*-------------------------------------------------------------------------------------------*/
  Reinicio();
}
```

Figura 54. Código incluido en el void setup()

La siguiente parte del código, la incorporada en *void loop()*, es la incluida en la parte del programa que se encuentra en constante ejecución, cuyo funcionamiento se basa en detectar si existe algún flujo de información en el módulo *bluetooth*. En caso afirmativo, lee el valor y lo almacena en una variable la cual, según sea su valor, permite que la placa realice una función u otra gracias a la estructura *switch () case:*.

Esta variable leída no es más que el *String* enviado por la aplicación basada en *Android* (ver *4.2.5 MainActivity.java*). Al recibir esta variable, el programa ejecuta constantemente una parte del código mientras no vuelva a detectar que haya flujo de información en el módulo.

Para cada caso de *String* enviado, existe una función *do{} while()*, por lo que solo se va a mostrar un caso como ejemplo. Además, según el caso, las funciones llamadas en cada uno varían en función de la tarea a realizar.

```
void loop() {

  vel = constrain(vel, 2, 6);

  if (BT.available()) {
    data = BT.read();

    switch (data) {

      /*-----SERVOMOTOR 1-----*/
      /*-----SERVOMOTOR 1 - MOVIMIENTO DERECHA-----*/
      case 'A':
        do {
          Motor1();
          Serial.write(data);
          delay(30);
        } while (!BT.available());
        break;
```

Figura 55. Ejemplo de código incluido en void loop()

También limita el valor de una variable cuya utilidad se verá más adelante.

Ya fuera de la parte del código correspondiente al *void loop()*, se encuentran las demás funciones encargadas de realizar las distintas tareas deseadas, desde el movimiento de cada uno de los servomotores, hasta la activación de los *joysticks* analógicos e incluso la función de reinicio.

Las funciones encargadas de realizar los giros de los servomotores son muy similares debido a que todos ellos funcionan de la misma manera. La única variación de la que disponen es con respecto a los valores leídos por los *joysticks* analógicos y los límites en los giros de éstos para

no provocar ningún problema en el movimiento del brazo. Por este motivo, además, solo se va a comentar el diseño de la función de uno de los motores, *Motor1()* en este caso.

Esta función evalúa el valor leído por la entrada analógica correspondiente a uno de los ejes de uno de los *joysticks* o al valor recibido por la conexión *bluetooth*. Según estos valores, aumenta o disminuye el valor de la variable que posteriormente se escribe en el servomotor, de forma que, al evaluarlo constantemente, aumenta o disminuye constantemente, hasta unos límites establecidos, por lo que sobrescribe continuamente la cantidad de gira del servomotor. Es necesario realizarlo de esta forma debido a la naturaleza de los servomotores (ver *3.1.5 Servomotores Microservo SG90 9g*).

```
void Motor1 () {

  if (valorX1 < 503 || data == 'A') grados1 -= 1 * vel;
  else if (valorX1 > 543 || data == 'B') grados1 += 1 * vel;

  grados1 = constrain(grados1, 20, 160);

  motor1.write (grados1); // ENVIAR LOS GRADOS AL SERVO 1
}
```

Figura 56. Función Motor1()

Al cálculo de la variable enviada a los servomotores para su giro se le ha añadido otra variable que la multiplica, a modo de cambio en la velocidad de éstos.

Para poder realizar las lecturas analógicas y digitales de los *joysticks*, es necesario tener variables donde guardar la lectura de éstos. Como además solo se requiere esta lectura cuando esté habilitado el modo de control mediante los *joysticks*, este código del programa, llamado función *Joysticks()*, solo será llamada en el caso que requiera la lectura de estas variables.

```
void Joysticks() {
  valorB1  = digitalRead (pinJB1);
  valorX1  = analogRead  (pinJX1);  // GUARDA LA LECTURA DEL PUERTO ANALOGICO A0 DEL EJE X
  valorY1  = analogRead  (pinJY1);  // GUARDA LA LECTURA DEL PUERTO ANALOGICO A1 DEL EJE Y
  valorX2  = analogRead  (pinJY2);  // GUARDA LA LECTURA DEL PUERTO ANALOGICO A2 DEL EJE X
  valorY2  = analogRead  (pinJX2);  // GUARDA LA LECTURA DEL PUERTO ANALOGICO A3 DEL EJE Y
}
```

Figura 57. Función Joysticks()

El pulsador del que se componen los *joysticks* funciona de tal manera que mientras se mantenga pulsado envía una señal digital al pin al que se encuentre conectado. Para modificar esto de forma que envíe la señal una sola vez al pulsarlo, es necesario implementar una función adicional, llamada *CambioEtado(p)*.

La función diseñada consiste en guardar el estado actual de la lectura del pin digital, para después evaluar si ha cambiado con respecto a su estado anterior. El cambio de estado se produce cada vez que el pulsador se acciona y se suelta, de forma que hay que implementar un contador para coger solo los casos en los que se pulsa el botón. Para esto, se calcula el resto de la división entre dos, de forma que, si es distinto de uno, la función devuelve *true*, y si es uno devuelve *false*.

```
boolean CambioEstado(int p) {
  estadoActual = digitalRead(p);
  if (estadoAnterior != estadoActual)
  {
    contador++;
    int validarPar = contador % 2;
    if (validarPar != 1)
    {
      estadoAnterior = estadoActual;
      return true;
    } else {
      estadoAnterior = estadoActual;
      return false;
    }

  } else return false;
}
```

Figura 58. Función CambioEstado(p)

El pulsador del *joystick* izquierdo (J2) se ha utilizado para el cambio de velocidad de giro de los servomotores. Para esto se ha creado otra función, *Velocidad()*, que devuelve una variable, que es la utilizada en las funciones que realizan el giro de los servomotores. Esta función evalúa el pulsador del *joystick*, gracias a la función *CambioEstado(p)*, de forma que al pulsarlo aumenta el valor de la variable a devolver. El valor predeterminado de esta variable es dos, aumenta de dos en dos al accionar el pulsador y, al tener un valor mayor que seis, vuelve a valer dos, de forma que el movimiento de los servomotores consta de tres velocidades diferentes.

```
float Velocidad() {

  if (CambioEstado(pinJB2)) vel += 2;
  if (vel > 6) vel = 2;

  return vel;
}
```

Figura 59. Función Velocidad()

La última función diseñada es la función *Reinicio()*, la cual, como su propio nombre indica, es la encargada de llevar al brazo robótico a su posición inicial. Esta función se llama en el caso de que estén habilitados los *joysticks* y se accione el pulsador del *joystick* derecho (J2) o si recibe su *String* correspondiente. En ambos casos, sobrescribe a un valor predeterminado las variables asociadas a los grados de giro de los servomotores y les escribe a éstos esos nuevos valores.

```
void Reinicio() {

  if (valorB1 == false || data == 'O')  {
    grados1 = 160;    // S1
    grados2 = 60;     // S2
    grados3 = 20;     // S3
    grados4 = 100;    // S4

    motor1.write(grados1);
    motor2.write(grados2);
    motor3.write(grados3);
    motor4.write(grados4);
  }
}
```

Figura 60. Función Reinicio()

5. Bibliografía

[1] Android Developers. (2019). *Bluetooth | Android Developers*. [online] Available at: https://developer.android.com/guide/topics/connectivity/bluetooth?hl=es-419 [Accessed 9 Jul. 2019].

[2] Android Developers. (2019). *Bluetooth | Android Developers*. [online] Available at: https://developer.android.com/guide/topics/connectivity/bluetooth?hl=es-419 [Accessed 9 Jul. 2019].

[3] Android, C., Contenidos, I., PDF, C., Frecuentes, P., NRtfTree, L., JRtfTree, L. and FKScript, L. (2019). *Indice de Contenidos | sgoliver.net*. [online] sgoliver.net. Available at: http://www.sgoliver.net/blog/curso-de-programacion-android/indice-de-contenidos/ [Accessed 9 Jul. 2019].

[4] Consumer.huawei.com. (2019). *HUAWEI P9 lite | Smartphones | HUAWEI España*. [online] Available at: https://consumer.huawei.com/es/phones/p9-lite/ [Accessed 9 Jul. 2019].

[5] Forum.arduino.cc. (2019). *Arduino Sensor Shield v5 (APC220) manual*. [online] Available at: https://forum.arduino.cc/index.php?topic=229646.0 [Accessed 9 Jul. 2019].

[6] García, D. (2019). *Bluetooth (I): Activando y desactivando el Bluetooth en Android*. [online] Let's code something up!. Available at: https://danielggarcia.wordpress.com/2013/10/19/bluetooth-i-activando-y-desactivando-el-bluetooth-en-android/ [Accessed 9 Jul. 2019].

[7] Internacional, A. (2019). [online] Edutech.atlantistelecom.com. Available at: https://edutech.atlantistelecom.com/post/ebotics-arm-robot-recursos-online-200.php [Accessed 9 Jul. 2019].

[8] Naylampmechatronics.com. (2019). *Configuración del módulo bluetooth HC-05 usando comandos AT*. [online] Available at: https://naylampmechatronics.com/blog/24_configuracion-del-modulo-bluetooth-hc-05-usa.html [Accessed 9 Jul. 2019].

[9] Pérez, E. (2019). *Cómo instalar el Android SDK y para qué nos sirve*. [online] Xatakandroid.com. Available at: https://www.xatakandroid.com/programacion-android/como-instalar-el-android-sdk-y-para-que-nos-sirve [Accessed 9 Jul. 2019].

[10] Prometec.net. (2019). *El módulo BlueTooth HC-05 | Tienda y Tutoriales Arduino*. [online] Available at: https://www.prometec.net/bt-hc05/ [Accessed 9 Jul. 2019].

[11] Prometec.net. (2019). *Indice de tutoriales Arduino | Tienda y Tutoriales Arduino*. [online] Available at: https://www.prometec.net/indice-tutoriales/ [Accessed 9 Jul. 2019].

[12] Prometec.net. (2019). *Módulo BlueTooth HC-06 | Tienda y Tutoriales Arduino*. [online] Available at: https://www.prometec.net/bt-hc06/ [Accessed 9 Jul. 2019].

[13] Rodriguez, J., juarez, d. and Cinjordiz, C. (2019). *Arduino Uno R3, tutorial especificaciones electrónicas y programación.*. [online] infootec.net. Available at: https://www.infootec.net/arduino/#1--Descripcion-y-caracteristicas-tecnicas [Accessed 9 Jul. 2019].

[14] v5, S. (2019). *Sensor Shield v5 - Naylamp Mechatronics - Perú*. [online] Naylamp Mechatronics - Perú. Available at: https://naylampmechatronics.com/arduino-shields/109-sensor-shield-v5.html [Accessed 9 Jul. 2019].

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO E IMPLEMENTACIÓN DEL CONTROL REMOTO DE UN BRAZO ROBÓTICO EDUCACIONAL UTILIZANDO DISPOSITIVOS BASADOS EN SISTEMA OPERATIVO ANDROID

DOCUMENTO 2.-

PRESUPUESTO DEL PROYECTO

REALIZADO POR

Daniel Uroz Franco

TUTORIZADO POR

Ricardo Pizá Fernández

FECHA: **Valencia, junio, 2019**

INDICE

1. Introducción

Este documento comprende la documentación detallada del presupuesto del proyecto realizado, con el fin de ver reflejado el coste total del proyecto.

Para tal efecto a continuación se muestran todos los gastos derivados de todos los recursos utilizados durante el desarrollo del proyecto:

- **Materiales:** todos los elementos físicos utilizado durante el desarrollo del proyecto.
- ***Software*:** los distintos programas implicados en la programación de las distintas aplicaciones.
- **Recursos humanos:** la mano de obra necesaria para la realización de las distintas tareas durante el proyecto

2. Materiales

Se incluyen todos los elementos físicos involucrados en el desarrollo del proyecto, ya sean los propios materiales que forman parte del brazo robótico o los necesarios para la implementación de la programación.

Descripción	Cantidad	Precio unitario (€)	Importe (€)
Kit Arm Robot de Ebotics	1	74,90	74,90
Módulo bluetooth HC-05	1	3,01	3,01
Cables puente hembra-hembra	6	0,021	0,13
Teléfono móvil Huawei P9 Lite	1	139,00	139,00
Ordenador portátil ASUS i5	1	250,00	250,00
		TOTAL	467,04

Tabla 1. Costes de materiales

3. Software

Se incluyen los programas utilizados para el desarrollo de los códigos implementados tanto en la placa electrónica como en la aplicación móvil.

Descripción	Cantidad	Precio unitario (€)	Importe (€)
Programa *Arduino IDE*	1	0,00	0,00
Programa *Android Studio*	1	0,00	0,00
		TOTAL	0,00

Tabla 2. Costes de software

4. Recursos humanos

Se incluye toda la mano de obra necesaria para la realización del proyecto, estando ésta formada por la intervención de un programador. El salario medio del programador pactado por los sindicatos es de 15,00€/h.

Descripción	Horas	Precio unitario (€/h)	Importe (€)
Análisis	15	15,00	225,00
Diseño	20	15,00	300,00
Implementación	45	15,00	675,00
Comprobación	30	15,00	450,00
		TOTAL	1650,00

Tabla 3. Costes de recursos humanos

5. Totales

Se suman todos costes calculados anteriormente.

Descripción	Importe (€)
Costes de materiales	467,04
Costes de *software*	0,00
Costes de recursos humanos	1650,00
TOTAL	2117,04

Tabla 4. Costes totales del proyecto

El presupuesto total del proyecto final es de **2117,04€**, repartidos en **467,04€** en concepto de materiales, **0.00€** en concepto de *software* y **1650,00€** en concepto de recursos humanos.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO E IMPLEMENTACIÓN DEL CONTROL REMOTO DE UN BRAZO ROBÓTICO EDUCACIONAL UTILIZANDO DISPOSITIVOS BASADOS EN SISTEMA OPERATIVO ANDROID

DOCUMENTO 3.-

MANUAL DE USUARIO

REALIZADO POR

Daniel Uroz Franco

TUTORIZADO POR

Ricardo Pizá Fernández

FECHA: **Valencia, junio, 2019**

INDICE

1. Introducción

En este documento va a contener toda la información necesaria para que cualquier usuario sea capaz de manejar fluidamente la aplicación móvil desarrollada y, por consiguiente, sea capaz de controlar a la perfección el brazo robótico *Arm Robot* de la marca *Ebotics* con el que está asociado.

Para ello se detallan paso a paso el funcionamiento de cada modo, como acceder a ellos y las funcionalidades de las que disponen, así como posibles fallos durante la conexión con el dispositivo *bluetooth* incorporado al robot.

2. Puesta en marcha del brazo robótico

Para encender el brazo robótico, es necesario alimentar a placa electrónica que lleva incorporada, puesto que es la encargada de realizar todos los movimientos posibles realizables por el brazo.

La alimentación se puede realizar de varias formas:

- Por conector USB tipo B conectándolo a un ordenador.
- Por entrada de *Jack* conectado a una fuente de 5V.

3. Pantalla principal

La aplicación dispone únicamente de una pantalla principal donde se muestran todas las funciones que es capaz de realizar.

Figura 1. Ventana principal

El botón “SALIR” siempre se encuentra habilitado y cierra la aplicación cuando se pulsa, finalizando la conexión *bluetooth* si estuviera activa y regresando a la pantalla principal del dispositivo móvil.

4. Realizar la conexión bluetooth

El primer paso es realizar la conexión con el dispositivo *bluetooth* que lleva incorporado el brazo robótico. Para ello, es necesario pulsar el botón “CONECTAR”.

En el caso de que el dispositivo móvil no tenga habilitado el modo bluetooth, se muestra mensaje en informando al usuario de que la aplicación desea activar este modo, dando a elegir al usuario si quiere permitírselo o no.

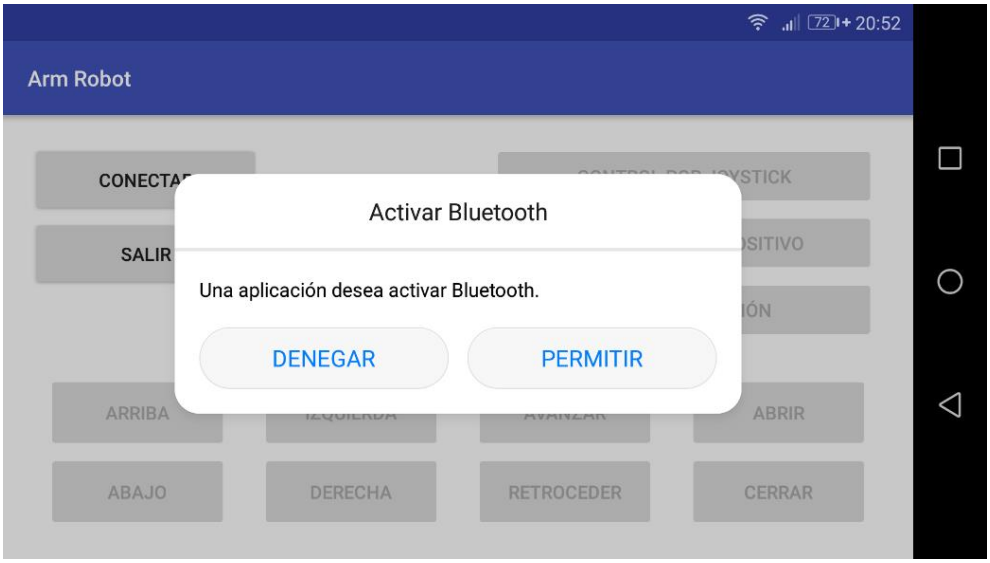

Figura 2. Activación del bluetooth

Si se permite la petición, se activa automática el modo *bluetooth* en el dispositivo móvil y se regresa a la pantalla principal, en cuyo caso hay que volver a presionar el botón “CONECTAR” si se desea volver a realizar el intento de conexión.

Durante este intento pueden existir problemas en la conexión. En este caso se muestra un pequeño mensaje con el texto “Fallo en el intento de conexión”. Este mensaje desaparece automáticamente a los pocos segundos. Por otra parte, la pantalla principal sigue activa para poder interactuar con ella.

Figura 3. Fallo en el intento de conexión

En el caso de que la conexión se haya realizado adecuadamente y sin ningún tipo de problema, se muestra otro pequeño mensaje, que también desaparece a los pocos segundos, con el mensaje “Bluetooth conectado”. Además, la pantalla principal recibe un pequeño cambio: se habilitan los botones correspondientes a los modos de funcionamiento y cambia el texto del botón “CONECTAR” por “DESCONECTAR”.

Figura 4. Bluetooth conectado

Figura 5. Pantalla con la conexión bluetooth realizada

Para cerrar la conexión *bluetooth*, basta con pulsar el botón "DESCONECTAR". Tras esto, se regresa a la pantalla principal (ver *Figura 1.Pantalla principal*).

Ahora es posible acceder a los distintos modos de control de los que dispone la aplicación: el "CONTROL POR JOYSTICK" y el "CONTROL POR DISPOSITIVO".

5.Control por joystick

Este modo es el que permite manejar el brazo robótico gracias a los *joysticks* analógicos que lleva incorporados a la periferia de la placa.

Para acceder a este modo, basta con pulsar el botón "CONTROL POR JOYSTICK".

Figura 6. Modo control por joystick

Durante este modo, la pantalla resultará como se muestra en la *Figura 6. Modo control por joystick*, además de habilitar los pines analógicos que realizan las lecturas de los ejes de los *joysticks*. El texto del botón "CONTROL POR JOYSTICK" se modifica por "DESACTIVAR MODO", lo que permite volver a deshabilitar estos pines y regresar a la pantalla mostrada en la *Figura 5. Pantalla con la conexión bluetooth realizada*.

Cada *joystick* dispone de dos ejes en los que se puede mover, eje x y eje y, además de un pulsador. Conociendo esto, la función de cada uno de estas posibilidades es la mostrada en la *Figura 7. Instrucciones del control por joystick*.

JOYSTICK	ACCIÓN	FUNCIÓN
Izquierdo (J1)	Mover eje X	Rotar a derecha e izquierda
	Mover eje Y	Mover hacia delante y hacia atrás
	Pulsador	Reiniciar posición
Derecho (J2)	Mover eje X	Abrir y cerrar pinza
	Mover eje Y	Mover hacia arriba y hacia abajo
	Pulsador	Cambiar velocidad

Figura 7. Instrucciones del control por joystick

6. Control por dispositivo

El otro modo del que dispone la aplicación es el modo remoto o "CONTROL POR DISPOSITIVO". Este modo se activa al pulsar el botón con el mismo nombre, de forma que la pantalla se modifica al habilitar los botones necesarios para realizar el control con el dispositivo móvil, los situados en la parte inferior de la aplicación, y deshabilitar el botón correspondiente al otro modo. También cambia el texto de el botón que activa este modo a "DESACTIVAR MODO" para poder salir de éste si se desea.

Figura 8. Modo control por dispositivo

Durante este modo, el control del brazo robótico se realiza exclusivamente desde el dispositivo móvil mediante los botones que indican el movimiento que se desea realizar, siendo estos: los movimientos hacia arriba, hacia abajo, hacia la derecha y hacia la izquierda; la rotación a derecha e izquierda; y la apertura y el cierre de la pinza.

Para acabar, este modo habilita un botón con el texto “REINICICAR POSICIÓN”, que permite llevar la posición del brazo robótico a una predeterminada.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO E IMPLEMENTACIÓN DEL CONTROL REMOTO DE UN BRAZO ROBÓTICO EDUCACIONAL UTILIZANDO DISPOSITIVOS BASADOS EN SISTEMA OPERATIVO ANDROID

DOCUMENTO 4.-

CÓDIGO DE LA PROGRAMACIÓN

REALIZADO POR

Daniel Uroz Franco

TUTORIZADO POR

Ricardo Pizá Fernández

FECHA: **Valencia, junio, 2019**

INDICE

1. ANDROID STUDIO

1.1.activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="18dp"
    tools:context=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="203dp"
        android:orientation="horizontal">

        <LinearLayout
            android:layout_width="150dp"
            android:layout_height="wrap_content"
            android:orientation="vertical">

            <Button
                android:id="@+id/btnConDesc"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:onClick="conexion_desconexion"
                android:text="Conectar"
                android:textOff="Conectar"
                android:textOn="Desconectar" />

            <Button
                android:id="@+id/btnSalir"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:onClick="salir"
                android:text="Salir" />

        </LinearLayout>

        <LinearLayout
            android:layout_width="250dp"
            android:layout_height="165dp"
            android:layout_marginStart="150dp"
            android:orientation="vertical">

            <Button
                android:id="@+id/btnJoy"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:enabled="false"
                android:onClick="modo_joystick"
                android:text="CONTROL POR JOYSTICK" />
```

```
            <Button
                android:id="@+id/btnDis"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:enabled="false"
                android:onClick="modo_dispositivo"
                android:text="CONTROL POR DISPOSITIVO" />

            <Button
                android:id="@+id/btnRei"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:enabled="false"
                android:onClick="reinicio"
                android:text="REINICIAR POSICIÓN" />
        </LinearLayout>

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:baselineAligned="false"
        android:orientation="horizontal">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="0.25"
            android:orientation="vertical">

            <Button
                android:id="@+id/btnArr"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:layout_marginRight="10dp"
                android:layout_weight="1"
                android:enabled="false"
                android:text="ARRIBA" />

            <Button
                android:id="@+id/btnAba"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:layout_marginRight="10dp"
                android:layout_weight="1"
                android:enabled="false"
                android:text="ABAJO" />
        </LinearLayout>

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_weight="0.25"
            android:orientation="vertical">
```

```
        <Button
            android:id="@+id/btnIzq"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:enabled="false"
            android:text="IZQUIERDA" />

        <Button
            android:id="@+id/btnDer"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:enabled="false"
            android:text="DERECHA" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="0.25"
        android:orientation="vertical">

        <Button
            android:id="@+id/btnAva"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:enabled="false"
            android:text="AVANZAR" />

        <Button
            android:id="@+id/btnRet"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
            android:layout_marginRight="10dp"
            android:layout_weight="1"
            android:enabled="false"
            android:text="RETROCEDER" />

    </LinearLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="0.25"
        android:orientation="vertical">

        <Button
            android:id="@+id/btnAbr"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginLeft="10dp"
```

```
                android:layout_marginRight="10dp"
                android:layout_weight="1"
                android:enabled="false"
                android:text="ABRIR" />

            <Button
                android:id="@+id/btnCer"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginLeft="10dp"
                android:layout_marginRight="10dp"
                android:layout_weight="1"
                android:enabled="false"
                android:text="CERRAR" />
        </LinearLayout>

    </LinearLayout>

</LinearLayout>
```

1.2. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.zoruda.proyectotfg">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Arm Robot"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
            android:screenOrientation="landscape">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

1.3. MainActivity.java

```
package com.zoruda.proyectotfg;

import java.io.IOException;
import java.io.OutputStream;
import java.lang.String;
import java.util.Set;
import java.util.UUID;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {


    /*-----ACTIVAR-----
    -------OPCIONES--*/
    BluetoothAdapter mAdapter;
    BluetoothSocket mSocket;
    BluetoothDevice mDevice;

    OutputStream mOutpoutStream;


    /*-----INICIALIZACION-----
    -------VARIABLES--------*/
    Button mBJoy = null;
    Button mBDis = null;
    Button mBArr = null;
    Button mBAba = null;
    Button mBDer = null;
    Button mBIzq = null;
    Button mBAva = null;
    Button mBRet = null;
    Button mBCer = null;
    Button mBAbr = null;
    Button mBRei = null;

    String address = null;

    UUID = UUID.fromString("00001101-0000-1000-8000-00805F9B34FB");

    boolean a = true;
    boolean b = true;
    boolean c = true;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
/*-----ASIGNACIÓN-----
-------DE BOTONES---*/
mBJoy = findViewById(R.id.btnJoy);
mBDis = findViewById(R.id.btnDis);
mBArr = findViewById(R.id.btnArr);
mBAba = findViewById(R.id.btnAba);
mBDer = findViewById(R.id.btnDer);
mBIzq = findViewById(R.id.btnIzq);
mBAva = findViewById(R.id.btnAva);
mBRet = findViewById(R.id.btnRet);
mBCer = findViewById(R.id.btnCer);
mBAbr = findViewById(R.id.btnAbr);
mBRei = findViewById(R.id.btnRei);


/*-----SERVOMOTOR 1-----*/

/*-----MOVIMIENTO DERECHA-----*/
mBDer.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            sendBT("A");
        }
        if (event.getAction() == MotionEvent.ACTION_UP) {
            sendBT("C");
        }
        return true;
    }
});

/*-----MOVIMIENTO IZQUIERDA-----*/
mBIzq.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            sendBT("B");
        }
        if (event.getAction() == MotionEvent.ACTION_UP) {
            sendBT("C");
        }
        return true;
    }
});


/*-----SERVOMOTOR 2-----*/

/*-----MOVIMIENTO AVANZAR-----*/
mBAva.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        if (event.getAction() == MotionEvent.ACTION_DOWN) {
            sendBT("D");
        }
        if (event.getAction() == MotionEvent.ACTION_UP) {
            sendBT("F");
        }
        return true;
```

```
        }
    });

    /*-----MOVIMIENTO RETROCEDER-----*/
    mBRet.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_DOWN) {
                sendBT("E");
            }
            if (event.getAction() == MotionEvent.ACTION_UP) {
                sendBT("F");
            }
            return true;
        }
    });


    /*-----SERVOMOTOR 3-----*/

    /*-----MOVIMIENTO ARRIBA-----*/
    mBArr.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_DOWN) {
                sendBT("G");
            }
            if (event.getAction() == MotionEvent.ACTION_UP) {
                sendBT("I");
            }
            return true;
        }
    });

    /*-----MOVIMIENTO ABAJO-----*/
    mBAba.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_DOWN) {
                sendBT("H");
            }
            if (event.getAction() == MotionEvent.ACTION_UP) {
                sendBT("I");
            }
            return true;
        }
    });


    /*-----SERVOMOTOR 4-----*/

    /*-----MOVIMIENTO CERRAR PINZA-----*/
    mBCer.setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            if (event.getAction() == MotionEvent.ACTION_DOWN) {
                sendBT("J");
            }
            if (event.getAction() == MotionEvent.ACTION_UP) {
                sendBT("L");
            }
```

```
                return true;
            }
        });

        /*-----MOVIMIENTO ABRIR PINZA-----*/
        mBAbr.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                if (event.getAction() == MotionEvent.ACTION_DOWN) {
                    sendBT("K");
                }
                if (event.getAction() == MotionEvent.ACTION_UP) {
                    sendBT("L");
                }
                return true;
            }
        });
    }


    /*-----FUNCIÓN ENCARGADA-----
    -------DE CONEXIÓN-----------
    -------POR BLUETOOTH-------*/
    public void startBT() throws IOException {

        mAdapter = BluetoothAdapter.getDefaultAdapter();

        if (mAdapter == null) {
            showMessage("Este dispositivo no dispone de conexión
Bluetooth", Toast.LENGTH_LONG);
            return;
        }

        if (!mAdapter.isEnabled()) {
            Intent enableBluetooth = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBluetooth, 1);
        }

        Set<BluetoothDevice> pairedDevices =
mAdapter.getBondedDevices();

        if (pairedDevices.size() > 0) {
            for (BluetoothDevice device : pairedDevices) address =
"98:D3:32:30:F7:5F";
        }

        try {
            mDevice = mAdapter.getRemoteDevice(address);
            mSocket =
mDevice.createInsecureRfcommSocketToServiceRecord(uuid);
            mSocket.connect();
            mOutpoutStream = mSocket.getOutputStream();
            mBDis.setEnabled(true);
            mBJoy.setEnabled(true);
            showMessage("Bluetooth conectado", Toast.LENGTH_SHORT);
            c = true;
        } catch (IOException e) {
            showMessage("Fallo en el intento de conexión",
Toast.LENGTH_SHORT);
            c = false;
```

```
        }
    }


    /*-----FUNCIÓN ENCARGADA-----
    -------DE DESCONECTAR-------
    -------EL BLUETOOTH-------*/
    public void endBT() throws IOException {
        mOutpoutStream.close();
        mSocket.close();
        c = false;

        mBDis.setEnabled(false);
        mBJoy.setEnabled(false);
        mBArr.setEnabled(false);
        mBAba.setEnabled(false);
        mBDer.setEnabled(false);
        mBIzq.setEnabled(false);
        mBAva.setEnabled(false);
        mBRet.setEnabled(false);
        mBAbr.setEnabled(false);
        mBCer.setEnabled(false);
        mBRei.setEnabled(false);
    }


    /*-----FUNCIÓN ENCARGADA-----
    -------DE MOSTRAR TOASTS---*/
    public void showMessage(String msg, int time) {
        Toast = Toast.makeText(MainActivity.this, msg, time);
        toast.show();
    }


    /*-----FUNCIÓN ENCARGADA-----
    -------DE ENVIAR LA----------
    -------INFORMACIÓN----------
    -------POR BLUETOOTH-------*/
    public void sendBT(String i) {
        try {
            if (mSocket != null) {
                mOutpoutStream.write(i.toString().getBytes());
            }
        } catch (Exception e) {
        }
    }


    /*-----FUNCIÓN ENCARGADA-----
    -------DE ACTIVAR------------
    -------MODO DISPOSITIVO----*/
    public void modo_dispositivo(View v) {
        if (a) {
            mBDis.setText("DESACTIVAR MODO");
            mBArr.setEnabled(a);
            mBAba.setEnabled(a);
            mBDer.setEnabled(a);
            mBIzq.setEnabled(a);
            mBAva.setEnabled(a);
            mBRet.setEnabled(a);
            mBCer.setEnabled(a);
```

```
            mBAbr.setEnabled(a);
            mBRei.setEnabled(a);
            mBJoy.setEnabled(!a);
        } else {
            mBDis.setText("CONTROL POR DISPOSITIVO");
            mBArr.setEnabled(a);
            mBAba.setEnabled(a);
            mBDer.setEnabled(a);
            mBIzq.setEnabled(a);
            mBAva.setEnabled(a);
            mBRet.setEnabled(a);
            mBCer.setEnabled(a);
            mBAbr.setEnabled(a);
            mBRei.setEnabled(a);
            mBJoy.setEnabled(!a);
        }
        a = !a;
    }


    /*-----FUNCIÓN ENCARGADA-----
    -------DE ACTIVAR------------
    -------MODO JOYSTICK-------*/
    public void modo_joystick(View v) {
        if (b) {
            mBJoy.setText("DESACTIVAR MODO");
            mBDis.setEnabled(!b);
            sendBT("M");
        } else {
            mBJoy.setText("CONTROL POR JOYSTICK");
            mBDis.setEnabled(!b);
            sendBT("N");
        }
        b = !b;
    }

    /*-----FUNCIÓN ENCARGADA-----
    -------DE ACTIVAR REINICIO-*/
    public void reinicio(View v) {
        sendBT("O");
    }


    /*-----FUNCIÓN ENCARGADA-----
    -------DE SALIR DE-----------
    -------LA APLICACIÓN-------*/
    public void salir(View v) throws IOException {
        endBT();
        finish();
    }


    /*-----FUNCIÓN ENCARGADA-----
    -------DE ACTIVAR Y----------
    -------DESACTIVAR------------
    -------LA CONEXIÓN-----------
    -------BLUETOOTH-----------*/
    public void conexion_desconexion(View v) {
        a = true;
        b = true;
        if (mSocket != null && mSocket.isConnected()) {
```

```
                try {
                    endBT();
                } catch (IOException e) {
                    showMessage("Fallo en la desconexión",
Toast.LENGTH_SHORT);
                }
            } else {
                try {
                    startBT();
                } catch (IOException e) {
                    showMessage("Fallo en la conexión",
Toast.LENGTH_SHORT);
                }
            }
            if (c == true) {
                ((Button) v).setText("Desconectar");
            } else {
                ((Button) v).setText("Conectar");
            }

        }
}
```

2.ARDUINO IDE

2.1. ConexionBluetooth.ino

```
#include <SoftwareSerial.h>

SoftwareSerial BT1(9, 10); //RX, TX

void setup()
   {
       Serial.begin(9600);
       Serial.println("Itroduce comandos AT:");
       BT1.begin(9600);
   }

void loop()
   {
       if (BT1.available())
           Serial.write(BT1.read());

       if (Serial.available())
           BT1.write(Serial.read());
}
```

2.2. CalibracionJoysticks.ino

```
#include <Servo.h>

int valorX = 0; // VARIABLE DE LECTURA DEL EJE X
int pinJX = A0; // EJE X CONECTADO AL PIN ANALOGICO A0

void setup() {
Serial.begin (9600); // ACTIVAR COMUNICACIÓN POR PUERTO SERIE
}

void loop() {

valorX = analogRead (pinJX);
Serial.print ("X: ");
Serial.println (valorX); // MOSTRAR POR PANTALLA LOS VALORES DEL EJE X
DE 0 A 1023
}
```

2.3. ArmRobot.ino

```
#include <Servo.h>
#include <SoftwareSerial.h>

int valorX1 = 0;       // LECTURA DEL EJE X de J1
int valorY1 = 0;       // LECTURA DEL EJE Y de J1
int valorX2 = 0;       // LECTURA DEL EJE X de J2
int valorY2 = 0;       // LECTURA DEL EJE Y de J2
bool valorB1 = false; // LECTURA DEL PULSADOR de J1
bool valorB2 = false; // LECTURA DEL PULSADOR de J2
/*-----------------------------------------------------------------*/
int pinJX1 = A0; // PIN ANALOGICO A1 DEL EJE X de J1
int pinJY1 = A1; // PIN ANALOGICO A0 DEL EJE Y de J1
int pinJB1 = 11; // PIN DIGITAL 11 DEL PULSADOR de J1
int pinJX2 = A2; // PIN ANALOGICO A2 DEL EJE X de J2
int pinJY2 = A3; // PIN ANALOGICO A3 DEL EJE Y de J2
int pinJB2 = 3;  // PIN DIGITAL 3 DEL PULSADOR de J2
/*-----------------------------------------------------------------*/
Servo motor1; // DECLARAR S1 -- SERVO MOTOR 1
Servo motor2; // DECLARAR S2 -- SERVO MOTOR 2
Servo motor3; // DECLARAR S3 -- SERVO MOTOR 3
Servo motor4; // DECLARAR S4 -- SERVO MOTOR 4
/*-----------------------------------------------------------------*/
int grados1 = 160; // GRADOS DEL S1
int grados2 = 60;  // GRADOS DEL S2
int grados3 = 20;  // GRADOS DEL S3
int grados4 = 90;  // GRADOS DEL S4
/*-----------------------------------------------------------------*/
SoftwareSerial BT(9, 10); // DECLARAR DISPOSITIVO BLUETOOTH HC-05(RX,
TX)
/*-----------------------------------------------------------------*/
int vel = 2;
```

```
bool estadoAnterior = false;
bool estadoActual;
int contador = 0;
char data;

void setup() {
  /*-----------------------------------------------------------------*/
  BT.begin(9600); //INICIAR BLUETOOTH A 9600 BAUD
  Serial.begin(9600);
  /*-----------------------------------------------------------------*/
  motor1.attach (8);  // PIN DIGITAL PWM 8 DONDE ESTÁ CONECTADO EL S1
  motor2.attach (2);  // PIN DIGITAL PWM 1 DONDE ESTÁ CONECTADO EL S2
  motor3.attach (13); // PIN DIGITAL PWM 13 DONDE ESTÁ CONECTADO EL S3
  motor4.attach (6);  // PIN DIGITAL PWM 6 DONDE ESTÁ CONECTADO EL S4
  /*-----------------------------------------------------------------*/
  pinMode (pinJB1, INPUT_PULLUP);  // PIN DIGITAL 11 CON CONFIGURACION
ENTRADA Y RESISTENCIA PULL UP
  pinMode (pinJB2, INPUT_PULLUP);  // PIN DIGITAL 3 CON CONFIGURACION
ENTRADA Y RESISTENCIA PULL UP
  /*-----------------------------------------------------------------*/
  Reinicio();
}

/*-----CÓDIGO EN EJECUCIÓN CONSTANTE-----*/
void loop() {

  vel = constrain(vel, 2, 6);

  if (BT.available()) {
    data = BT.read();

    switch (data) {

      /*-----SERVOMOTOR 1-----*/
      /*-----SERVOMOTOR 1 - MOVIMIENTO DERECHA-----*/
      case 'A':
        do {
          Motor1();
          Serial.write(data);
          delay(30);
        } while (!BT.available());
        break;

      /*-----SERVOMOTOR 1 - MOVIMIENTO IZQUIERDA-----*/
      case 'B':
        do {
          Motor1();
          Serial.write(data);
          delay(30);
        } while (!BT.available());
        break;

      /*-----PARAR MOVIMIENTO SERVOMOTOR 1-----*/
      case 'C':
        Serial.write(data);
        delay (30);
        break;

        /*-----SERVOMOTOR 2-----*/
      /*-----SERVOMOTOR 2 - MOVIMIENTO ARRIBA-----*/
      case 'D':
```

```
  do {
    Motor2();
    Serial.write(data);
    delay(30);
  } while (!BT.available());
  break;

/*-----SERVOMOTOR 2 - MOVIMIENTO ABAJO-----*/
case 'E':
  do {
    Motor2();
    Serial.write(data);
    delay(30);
  } while (!BT.available());
  break;

/*-----PARAR MOVIMIENTO SERVOMOTOR 2-----*/
case 'F':
  Serial.write(data);
  delay (30);
  break;

/*-----SERVOMOTOR 3-----*/
/*-----SERVOMOTOR 3 - MOVIMIENTO AVANZAR-----*/
case 'G':
  do {
    Motor3();
    Serial.write(data);
    delay(30);
  } while (!BT.available());
  break;

/*-----SERVOMOTOR 3 - MOVIMIENTO RETROCEDER-----*/
case 'H':
  do {
    Motor3();
    Serial.write(data);
    delay(30);
  } while (!BT.available());
  break;

/*-----PARAR MOVIMIENTO SERVOMOTOR 3-----*/
case 'I':
  Serial.write(data);
  delay (30);
  break;

/*-----SERVOMOTOR 4-----*/
/*-----SERVOMOTOR 4 - MOVIMIENTO ABRIR-----*/
case 'J':
  do {
    Motor4();
    Serial.write(data);
    delay(30);
  } while (!BT.available());
  break;

/*-----SERVOMOTOR 4 - MOVIMIENTO CERRAR-----*/
case 'K':
  do {
    Motor4();
```

```
          Serial.write(data);
          delay(30);
        } while (!BT.available());
        break;

      /*-----PARAR SERVOMOTOR 4-----*/
      case 'L':
        Serial.write(data);
        delay (30);
        break;

       /*-----ENTRAR MODO JOYSTICK-----*/
      case 'M':
        do {
          vel = Velocidad();
          Joysticks();
          Motor1();
          Motor2();
          Motor3();
          Motor4();
          Reinicio();
          delay (30);
        } while (!BT.available());
        break;

      /*-----SALIR MODO JOYSITCK-----*/
      case 'N':
        Serial.write(data);
        delay (30);
        break;

      /*-----ACTIVAR REINICIO-----*/
      case 'O':
        do {
          Reinicio();
          Serial.write(data);
          delay (30);
        } while (!BT.available());
        break;
    }
  }
  delay (20);
}


/*-----FUNCIÓN REINICIO SERVOMOTORES-----*/
void Reinicio() {

  if (valorB1 == false || data == 'O')  {
    grados1 = 160;     // S1
    grados2 = 60;      // S2
    grados3 = 20;      // S3
    grados4 = 100;     // S4

    motor1.write(grados1);
    motor2.write(grados2);
    motor3.write(grados3);
    motor4.write(grados4);
  }
}
```

```
/*-----FUNCIÓN AJUSTE VELOCIDAD-----*/
float Velocidad() {

  if (CambioEstado(pinJB2)) vel += 2;
  if (vel > 6) vel = 2;

  return vel;
}


/*-----FUNCIÓN MOVIMIENTO SERVOMOTOR 1-----*/
void Motor1 () {

  if (valorX1 < 503 || data == 'A') grados1 -= 1 * vel;
  else if (valorX1 > 543 || data == 'B') grados1 += 1 * vel;

  grados1 = constrain(grados1, 20, 160);

  motor1.write (grados1); // ENVIAR LOS GRADOS AL SERVO 1
}


/*-----FUNCIÓN MOVIMIENTO SERVOMOTOR 2-----*/
void Motor2 () {

  if (valorY1 < 515 || data == 'D') grados2 += 1 * vel;
  else if (valorY1 > 555 || data == 'E') grados2 -= 1 * vel;

  grados2 = constrain(grados2, 60, 140);

  motor2.write (grados2); // ENVIAR LOS GRADOS AL SERVO 2
}


/*-----FUNCIÓN MOVIMIENTO SERVOMOTOR 3-----*/
void Motor3 () {

  if (valorX2 < 497 || data == 'G') grados3 += 1 * vel;
  else if (valorX2 > 537 || data == 'H') grados3 -= 1 * vel;

  grados3 = constrain(grados3, 20, 60);

  motor3.write (grados3); // ENVIAR LOS GRADOS AL SERVO 3

}


/*-----FUNCIÓN MOVIMIENTO SERVOMOTOR 4-----*/
void Motor4 () {

  if (valorY2 < 400 || data == 'J') grados4 += 2;
  else if (valorY2 > 600 || data == 'K') grados4 -= 2;

  grados4 = constrain(grados4, 60, 100);

  motor4.write (grados4); // ENVIAR LOS GRADOS AL SERVO 4
}


/*-----FUNCIÓN HABILITAR JOYSTICK-----*/
```

```
void Joysticks() {
  valorB1  = digitalRead (pinJB1);
  valorX1  = analogRead  (pinJX1);  // GUARDA LA LECTURA DEL PUERTO
ANALOGICO A0 DEL EJE X
  valorY1  = analogRead  (pinJY1);  // GUARDA LA LECTURA DEL PUERTO
ANALOGICO A1 DEL EJE Y
  valorX2  = analogRead  (pinJY2);  // GUARDA LA LECTURA DEL PUERTO
ANALOGICO A2 DEL EJE X
  valorY2  = analogRead  (pinJX2);  // GUARDA LA LECTURA DEL PUERTO
ANALOGICO A3 DEL EJE Y
}


/*-----FUNCIÓN PARA DETECTAR PULSO DE BAJADA-----*/
boolean CambioEstado(int p) {
  estadoActual = digitalRead(p);
  if (estadoAnterior != estadoActual)
  {
    contador++;
    int validarPar = contador % 2;
    if (validarPar != 1)
    {
      estadoAnterior = estadoActual;
      return true;
    } else {
      estadoAnterior = estadoActual;
      return false;
    }

  } else return false;
}
```