UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Comunicación inalámbrica entre microcontroladores para controlar un robot de forma remota

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática

AUTOR/A: Montesinos Berenguer, Daniel

Tutor/a: Rodríguez Ballester, Francisco

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

Comunicación inalámbrica entre microcontroladores para controlar un robot de forma remota.

Trabajo Fin de Grado

Grado en Ingeniería Electrónica Industrial y Automática.

Autor: Montesinos Berenguer, Daniel

Tutor: Rodríguez Ballester, Francisco

Curso Académico: 2021/2022

Índice General

Agradecimientos:

Este trabajo no habría sido posible sin el apoyo de mis familiares, a mis padres y a mi hermano les he de agradecer el haber hecho todo lo posible para brindarme la oportunidad de vivir esta etapa y la confianza incondicional que han tenido siempre en mí.

Agradecer a mis amistades, que pese a haber estado lejos han hecho que todo siga como siempre y a las que he conocido durante estos cuatro años, que han hecho que cada momento vivido tanto en el aula como fuera haya sido inolvidable.

No quisiera olvidarme del papel de los docentes que durante estos últimos años de grado han contribuido en mi formación, con mención especial a mi tutor, Francisco Rodríguez, que me ofreció este proyecto con el cual he disfrutado y aprendió a partes iguales y me ha guiado durante su realización.

Resumen

El principal objetivo de este proyecto es realizar el control de un robot móvil. El control se implementa mediante un sistema de dos microcontroladores, uno de ellos situado en el robot, al que se le conectan los diferentes sensores y actuadores y otro situado en el mando.

El control del robot se realiza mediante un *joystick* y una disposición de botones que permite navegar por un menú situado en una pantalla LCD incluida en el mando. Al seleccionar el modo de funcionamiento, conecta con el robot y permite su movimiento y control a través del mismo *joystick*.

La comunicación entre microcontroladores se realizará a través de comunicaciones inalámbricas y el canal serie. Se dispone de dispositivos receptores y emisores, conectados a microcontrolador, para realizar la transmisión de datos.

Finalmente, se implementa un control de seguridad, a través de los diferentes sensores, que impide la colisión del robot independientemente de la orden que se le envíe.

Palabras claves

Control. Microcontrolador. Comunicación. Inalámbrica.

ABSTRACT

The main objective of this project is to control a mobile robot. The control is implemented by a system of two microcontrollers, one of them located in the robot, to which the different sensors and actuators are connected, and the other one located in the controller.

The robot is controlled by means of a joystick and an arrangement of buttons that allows navigation through a menu located on an LCD screen included in the controller. When the operating mode is selected, it connects to the robot and allows its movement and control through the same joystick.

Communication between microcontrollers will be via wireless communications and the serial channel (UART). Receiver and transmitter devices, connected to the microcontroller, are available for data transmission.

Finally, a safety control is implemented, through the different sensors, which prevents the robot from colliding regardless of the order sent to it and the operating mode in which it is found.

Keywords

Control. Microcontroller. Communication. Wireless.

Resum

El principal objectiu d'este proyecte és realisar el control d'un robot mòvil. El control s'implementa per mig d'un sistema de dos microcontroladors, un d'ells situat en el robot, al que se li conecten els diferents sensors i actuadors i un atre situat en el mando.

El control del robot es realisa per mig d'un *joystick i una disposició de botons que permet navegar per un menú situat en una pantalla *LCD inclosa en el mando. En seleccionar el modo de funcionament, conecta en el robot i permet el seu moviment i control a través del mateix *joystick.

La comunicació entre microcontroladors es realisarà a través de comunicacions inalàmbriques i el canal serie. Es dispon de dispositius receptors i emissors, conectats a microcontrolador, per a realisar la transmissió de senyes.

Finalment, s'implementa un control de seguritat, a través dels diferents sensors, que impedix la colisió del robot independentment de l'orde que se li envie.

Paraules clau

Control. Microcontrolador. Comunicació. Inalàmbrica..

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

Comunicación inalámbrica entre microcontroladores para controlar un robot de forma remota.

1. MEMORIA

Autor: Montesinos Berenguer, Daniel

Tutor: Rodríguez Ballester, Francisco

Curso Académico: 2021/2022

Índice Memoria

Índice Figuras

Índice Tablas

1. Objeto

El objeto de este proyecto es la comunicación de dos microcontroladores de manera inalámbrica para el control remoto de un robot. Este dispondrá de diferentes modos de funcionamientos.

A la hora de elaborar este proyecto se definirá el tipo de protocolo de comunicación y cómo se llevará está a cabo. Además, se definirán los componentes tanto del robot como del mando. Por un lado, en lo relacionado al robot se seleccionará un kit analizando las características del microcontrolador que lo gobernará, el tipo de motor y controlador para el movimiento, el material a utilizar para el montaje y los diferentes sensores. Por otro lado, en cuanto al control, se elegirá otro kit poniendo énfasis en el microcontrolador, la interfaz y los periféricos que este utilizará.

Una vez seleccionados todos los componentes se realizarán dos códigos para, una vez establecida la comunicación, realizar el control.

2. Antecedentes

Hoy en día, los microcontroladores son una de las herramientas que más se utilizan, se pueden encontrar tanto en los procesos más sofisticados de la industria como en aplicaciones más simples como en un juguete infantil.

El campo de uso de los microcontroladores se expande a cualquier proceso que necesite de una automatización o un procesamiento de información. Debido a su gran uso es necesario familiarizarse con estos, la lectura de sus manuales y sus interfaces.

En la actualidad, los microcontroladores suelen ir incorporados en placas de circuito impreso (PCB) o en placas de desarrollo. En el primer caso, es necesario conocer las posibilidades que ofrece para así poder realizar el diseño óptimo y maximizar el funcionamiento de estos. En el segundo caso, saber interpretar la interfaz física que ofrecen las diferentes placas de desarrollo permite, además de la conexión adecuada de los diferentes periféricos, alargar la vida útil de estas.

Debido a la gran versatilidad y funcionalidad, los microcontroladores deben comunicarse con el resto de los elementos del sistema donde se encuentren para que este funcione. Por ello, además de comprender su funcionamiento y saber trabajar con ellos es imprescindible entender y utilizar tanto el medio (cable, inalámbrico), como el protocolo (CAN, MODBUS, I2C …) de comunicación que se ajuste mejor a cada aplicación.

Hoy en día, uno de los usos más extendidos de los microcontroladores es para el control remoto de robots en aplicaciones industriales, militares, quirúrgicas, etc. Esta gran variedad de funcionalidades que se les puede otorgar reafirma la idea de su importancia en la actualidad y en el futuro del sector tecnológico.

Los factores mencionados anteriormente motivan el desarrollo de este control remoto de un robot mediante la comunicación entre microcontroladores.

3. Estudio de necesidades, factores a considerar: limitaciones y condicionantes

El primer paso en el diseño y realización de un proyecto es plantear y definir las necesidades a las que se le irán dando solución durante el desarrollo del mismo. Este planteamiento sirve para tener una línea de trabajo concreta y bien definida para no desviarse del principal objetivo durante las distintas fases del desarrollo.

A continuación, se nombran y se explican las diferentes necesidades y características que se han de tener en consideración para el correcto funcionamiento final.

3.1 Comunicación inalámbrica

Este apartado es el más crítico del proyecto, debido a que el correcto funcionamiento del conjunto depende de que la comunicación inalámbrica sea robusta.

Por un lado, la solución inalámbrica que se elija debe de ser fiable. Esta solución debe garantizar una distancia máxima de transmisión que le otorgue al sistema un rango de funcionamiento amplio. Sin embargo, no será útil una solución que permita una gran distancia de funcionamiento entre emisor y receptor si la fiabilidad de conexión entre los elementos no es excelente.

Por lo tanto, la solución que se seleccione debe de encontrar la relación adecuada entre la distancia máxima a la que se pueden encontrar el emisor y el receptor y la fiabilidad y robustez de esta conexión.

3.2 Consumo energético y autonomía

El sistema tiene una orientación inalámbrica al estar pensado para que el robot funcione de manera independiente y por tanto no esté conectado a ninguna fuente de alimentación fija, además, debido a esta independencia del robot, es necesario pensar en un control que también sea capaz de funcionar con una batería.

El diseño del robot debe ser capaz de contener una batería de corriente continúa para su funcionamiento. Debido al tiempo finito de duración de estas, es necesario que microcontrolador, sensores, actuadores y módulo de comunicación, consuman la mínima potencia posible, de esta manera, se podrá alargar el tiempo de funcionamiento autónomo.

De manera análoga al diseño del robot, la elección de componentes para el control se ha de realizar teniendo presente intentar consumir la menor potencia posible.

Por último, es indispensable que ambos componentes del sistema tengan la capacidad de ser alimentados por fuentes fijas (ordenador, fuente de alimentación…) para así, facilitar su puesta en marcha y diseño.

3.3 Robustez

Un factor diferencial en el desarrollo de este proyecto es la robustez del sistema. Cada uno de los modos de funcionamiento ha de ser capaz de funcionar sin que suceda un error, y en el caso de que este se produzca, ha de ser capaz de gestionarlo sin que interfiera con el correcto funcionamiento del sistema.

Para ello, el sistema se diseñará teniendo en cuenta los posibles errores durante la ejecución. Debido a este planteamiento, se le otorgan herramientas al sistema para que evite los posibles errores, y en el caso de que estos sucedan los pueda solventar sin intervención.

3.4 Mejora continúa

Debido a la naturaleza tecnológica del proyecto, es necesario que diseño e implementación favorezcan futuras mejoras y modificaciones de este.

Teniendo en cuenta las futuras modificaciones es necesario que el diseño deje libre entradas y salidas de ambos microcontroladores para la futura conexión de sensores, actuadores y periféricos . Por otro lado, la programación y comunicación se debe realizar de una manera clara y modular, con la idea de que sea entendible y modificable.

Por último, la interfaz que se muestre en el control ha de ser intuitiva para que su uso y posible actualización en el futuro no suponga un inconveniente.

4. Planteamiento de soluciones alternativas y justificación de la solución adoptada

A la hora de plantear un proyecto, es necesario analizar y considerar la gran mayoría de las opciones que sean posibles de implementar, ya que, un correcto análisis permitirá disminuir la dificultad de la solución y aumentar la calidad de esta. El objetivo de este apartado es presentar las opciones que se han tenido en cuenta en los diferentes aspectos del proyecto y, tras este análisis, justificar el porqué de la solución adoptada, la cual será explicada en mayor detalle en el apartado 5.

4.1 Elección de la tecnología inalámbrica

La elección de la tecnología inalámbrica es la primera decisión que se ha de tomar, y, por tanto, esta decisión es el punto de partida para restringir las soluciones alternativas y ayudar a tomar la decisión final en los siguientes apartados. Teniendo en cuenta que ha de ser una tecnología fiable, robusta y que no aumente de manera drástica la dificultad de la implementación final, a continuación, se detallan tres de las tecnologías inalámbricas con un uso más extendido con el objetivo de determinar la que más se ajuste al proyecto.

4.1.1 WiFi

La tecnología inalámbrica con más usuarios en la actualidad es WiFi, cuyo logotipo se puede observar en la Figura 1. Esta tecnología da la posibilidad de la comunicación inalámbrica de dispositivos electrónicos, tales como ordenadores, videoconsolas, televisores, etc. Para la comunicación efectiva de tantos dispositivos diferentes, la tecnología WiFi se rige por un estándar para la conexión de los dispositivos anteriormente mencionados entre sí o bien con Internet.

Figura 1: Logotipo WiFi

El WiFi permite un gran ancho de banda, a cambio de consumir una gran cantidad de energía para poder realizar la transmisión, recepción y modulación de los datos. Esta característica permite que esta tecnología sea óptima para sistemas alimentados con fuentes fijas y de gran capacidad, pero hace que la implementación en

sistemas móviles que se alimentan a través de una batería sea más compleja. Este aumento de la dificultad se debe a que para alimentar los dispositivos que permiten la comunicación WiFi se necesita incluir baterías de tamaño considerable, del orden de 2000mAh. Por otro lado, esta tecnología opera en la banda de 2.4GHz, y esto permite que su alcance máximo ronde los 100m.

En resumen, esta tecnología permite intercambiar una cantidad de datos y a una velocidad mucho más elevada de las necesarias para esta aplicación. Sin embargo, la razón con más peso para que esta opción quede descartada es la gran cantidad de energía que consume. El tamaño de un sistema autónomo que utilice esta tecnología es mucho mayor que el idóneo para esta aplicación.

4.1.2 ZigBee

ZigBee nace con el objetivo de usarse en aplicaciones que necesiten de una comunicación segura, con una tasa baja de transmisión de datos y con una gran independencia energética, ya que el objetivo del diseño de esta tecnología es proporcionar un alto rendimiento a las aplicaciones donde el consumo de energía es importante.

La domótica es la principal área de implementación de ZigBee. Su funcionamiento típico es de una tipología de malla, aunque también se puede diseñar con tipología de árbol o estrella. En la Figura 2 se observa una topología de malla donde se muestran los 3 tipos de dispositivo que define ZigBee:

- **Dispositivo Final ZigBee (*end device*):** Estos dispositivos son los más simples que se encuentran en la red. Tienen la capacidad de comunicación con la red, pero no la capacidad de enrutar. Además, son capaces de entrar en modo de ahorro de energía para alargar la vida de la batería, volviendo al modo de funcionamiento normal únicamente cuando vayan a trabajar.
- **Coordinador:** Este dispositivo tiene la responsabilidad de crear la red, enrutar los diferentes paquetes y permitir las conexiones entrantes del resto de nodos de la red. Habitualmente, este dispositivo se integra junto a un módulo de conexión a Internet para enviar los datos que circulan por la red a un servidor central.
- **Router:** Su funcionamiento es análogo al coordinador, pero no tiene la capacidad de aceptar conexiones, únicamente tiene la capacidad para enrutar paquetes.

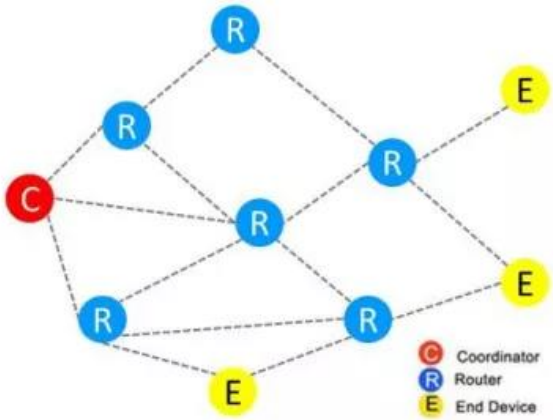

Figura 2: Estructura de una Red ZigBee

Replicando la estructura de la Figura 2, ZigBee permite crear redes de mallas escalables y versátiles. Estas redes pueden trabajar en las bandas de 2.4 GHz (utilizadas también por Bluetooth y WiFi), pero también puede trabajar en las bandas de 868MHz en Europa y 915MHz (Estados Unidos). Estas redes pueden estar conectadas hasta un máximo de 75 metros con una velocidad de transmisión de hasta 250 Kbps.

Como conclusión en cuanto a la tecnología ZigBee, sus características principales entran dentro de los deseado para esta aplicación, ya que su consumo de energía es bajo y su velocidad de transmisión es adecuada. Sin embargo, no se puede obviar la dificultad de su implementación debido a la complejidad de la creación y mantenimiento de la red, teniendo en cuenta que solo se van a conectar dos dispositivos.

4.1.3 Bluetooth Low Energy

Con el objetivo intentar compensar el gran consumo energético del WiFi y la complejidad de implementación de ZigBee, se considera el BLE (Bluetooth *Low Energy*). El BLE se define como una versión ligera del clásico Bluetooth.

Este tipo de Bluetooth permite la interconexión de un dispositivo central que envía los datos con uno periférico que los recibe a un máximo de 50m transmitiéndose un dato en un tiempo establecido: a menor tiempo, mayor consumo. La diferenciación de los BLE es que su consumo es considerablemente menor que los dispositivos Bluetooth clásicos y por tanto permite que la vida de las baterías que los alimenta funcione durante más tiempo. Esta tecnología funciona en frecuencias comprendidas entre 2.4000 y 2.4835 GHz.

La tecnología BLE ha dominado el campo de los proyectos DIY(*Do it Yourself* o hazlo tú mismo) debido a su bajo coste, su facilidad de implementación y la buena relación que tiene con los microcontroladores, ya que el módulo BLE es el encargado de enviar o recibir la información y el microcontrolador de procesarla.

Finalmente, se utilizará esta tecnología inalámbrica. El primer factor que se ha considerado para elegir BLE como tecnología es su bajo consumo, a continuación, la facilidad de implementación. Este último motivo viene apoyado por la cantidad de proyectos que ya se han realizado utilizando esta tecnología que proveerá a este proyecto de una gran fuente de documentación.

4.2 Tipo de dispositivo inalámbrico

Una vez elegida el tipo de comunicación inalámbrica será necesario estudiar las diferentes opciones que propone el mercado para implementar la comunicación Bluetooth tanto en el emisor como en el receptor. A lo largo de este apartado se analizarán los módulos cuyo uso está más extendido.

4.2.1 Seeed Studio 102080028

En la Figura 3 se puede observar el dispositivo Seeed Studio 102080028. Este dispositivo es un módulo de comunicación bluetooth desarrollado por el fabricante Seeed Studio.

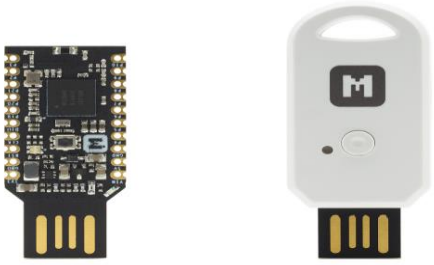

Figura 3: Dispositivo Seeed Studio 102080028

Este módulo dispone de un microprocesador ARM Cortex-M4F optimizado para sistemas que operan en bajo consumo. Por otro lado, también dispone de una memoria integrada de 1 MB de FLASH y 256 kB de RAM. Además, en la Figura 4 se muestra la asignación de pines del módulo.

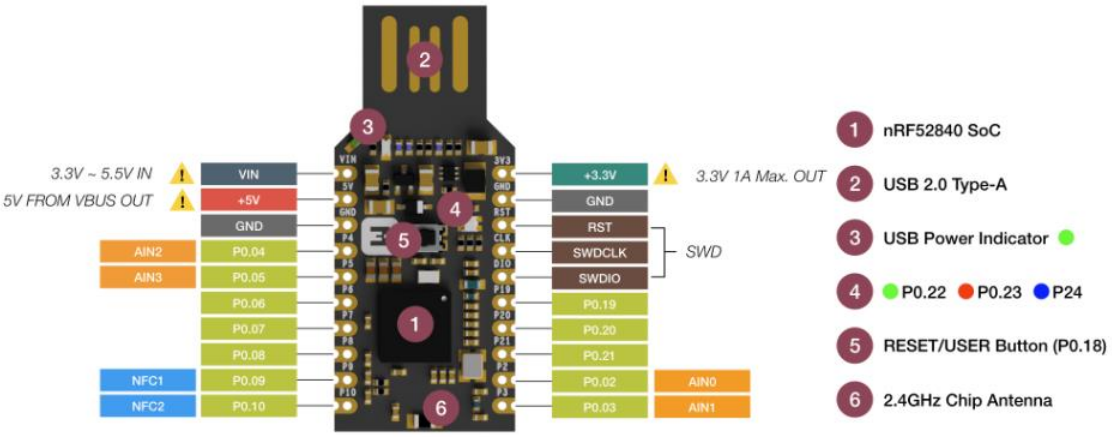

Figura 4: Asignación de pines del módulo Seeed Studio 102080028.

En los diferentes puntos de la Figura 4 se muestran los diferentes componentes que terminan de definir el módulo. El punto 1 se trata del microcontrolador explicado anteriormente, el punto 2 es un USB 2.0 tipo A, el punto 3 es un indicador LED que se enciende cuando el módulo está encendido, el punto 4 indica los diferentes LEDs junto a los que podemos encontrar las diferentes conexiones del *Serial Wire Debug* (SWD), utilizado para la programación de la familia ARM, en el punto 5 el botón de usuario y por último el 6 es la antena de comunicación de 2.4 GHz. Por otro lado, también se observa en la Figura 4 que el dispositivo tiene una entrada entre 3.3 y 5.5 V además de una interfaz de entrada-salida de 12 pines.

La funcionalidad de este módulo permite al usuario un gran abanico de posibilidades a la hora de implementar diferentes sistemas gracias a su microcontrolador ARM, sus 12 pines de interfaz entrada salida y su memoria, además de su antena ya incorporada. Sin embargo, se rechaza esta propuesta debido a que su interfaz en poco intuitiva y realizar las conexiones para una correcta comunicación con este dispositivo dificultaría en gran medida el proyecto sin otorgar una mejora considerable.

4.2.2 Bluno Bee SKU:TEL0073

En la Figura 5 se puede observar el dispositivo Bluno Bee SKU:TEL0073. Este dispositivo es un módulo de comunicación bluetooth desarrollado por DFRobot.

Figura 5: Dispositivo Bluno Bee SKU:TEL0073

Este módulo dispone del chip CC2540 de Texas Instruments, este componente es un microcontrolador con unidad de bluetooth integrada a la cual se le incorpora una antena enrutada en el propio módulo. A continuación, en la Figura 6 se muestra la asignación de pines del módulo y el resto de los componentes.

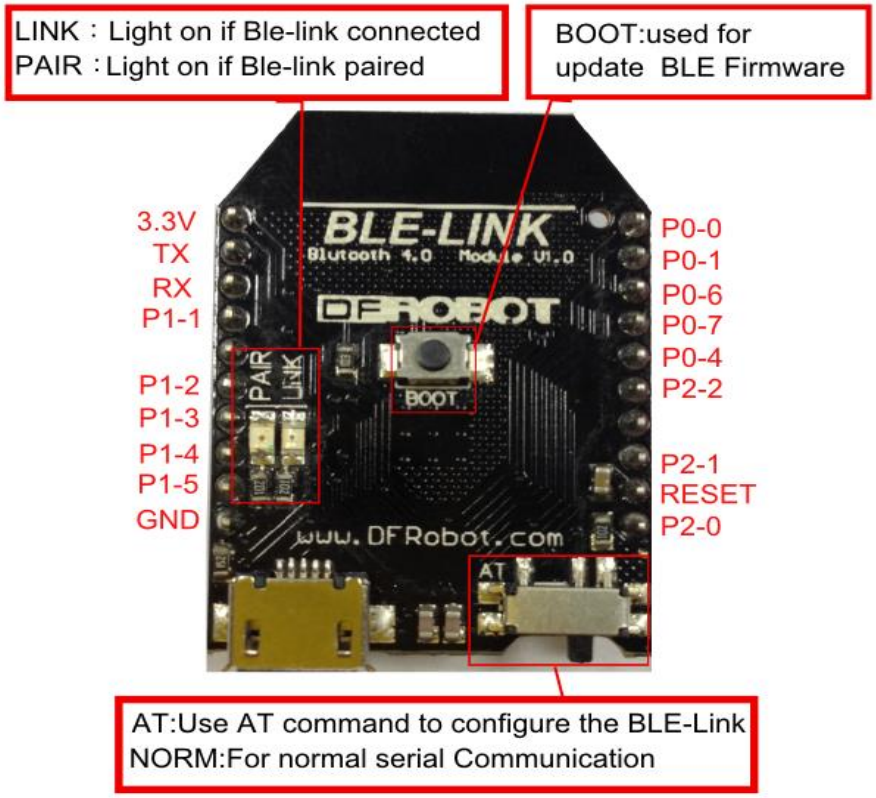

Figura 6: Asignación de pines del módulo Bluno Bee SKU:TEL0073

En la Figura 6 se muestra la funcionalidad tanto de los diferentes pines y componentes que se distribuyen en la superficie del componente. Se puede observar que el componente funciona a 3.3V, siendo su consumo medio de 10.6mA en

funcionamiento. Por otro lado, se puede observar que se implementa en el módulo una interfaz de entrada-salida que consta de 12 pines, además de tres extras, dos (Rx, Tx) para la transmisión de datos y uno para el *reset*. Además, el módulo implementa una entrada de alimentación, un interruptor para el modo de configuración y dos LEDs para indicar el modo en el que se encuentra el dispositivo (*Link*, *Pair*).

Este módulo no sólo es óptimo para la implementación de sistemas de diversos tipos, sino que además sus conexiones para la comunicación son simples y eficaces, utilizando el pin Rx para recibir la información y el pin Tx para enviarla. Sin embargo, como se observa en la Figura 6, este módulo dispone de muchas entradas y salidas que en este proyecto no se van a utilizar debido a que se conecta a un microcontrolador el cual es el encargado de gestionar las entradas y salidas y, por tanto, se descarta esta opción debido a que sería sobredimensionar el proyecto.

4.2.3 Módulos HC-05 y HC-06

En la Figura 7 se puede observar los dispositivos HC-05 y HC-06. El primero de ellos es el emisor y el segundo es el receptor.

Figura 7: Módulos HC-05 y HC-06

En primer lugar, se analiza el HC-05, este dispositivo utiliza el protocolo de Bluetooth estándar, transmitiendo la información con una velocidad entre 1 y 2 Mbps en una frecuencia entre 2.4 y 2.48 GHz. En la Figura 8 se observa la asignación de conexiones de los pines que incluye este módulo.

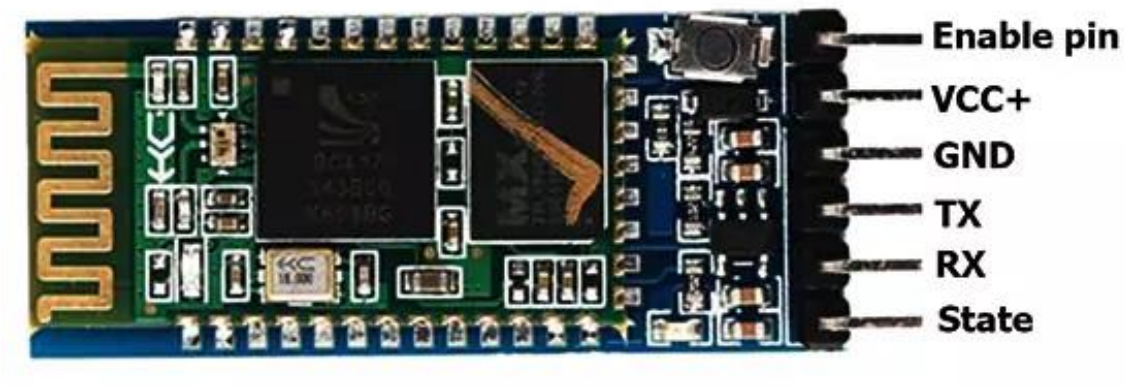

Figura 8: Asignación de pines de módulo HC 05

En Figura 8 se muestra la asignación de pines del dispositivo. En primer lugar, el Enable Pin, es el encargado de dar la información del si el módulo está funcionando en el modo de comandos AT o en el modo normal. A continuación, se encuentra el pin Vcc+, al cual se le asigna la alimentación del módulo. El rango de operación de este módulo es de +3.3 a +6 V, con una corriente de funcionamiento media de 30mA. En tercer lugar, se encuentra el pin GND, que irá conectado a la masa de la alimentación. Los siguientes dos pines Rx y Tx, son los encargados de conectarse para la comunicación, siendo el primero de estos el encargado de recibir la información y el segundo el de transmitirla. Por último, se encuentra el pin State, este pin está conectado al indicador led de la placa y nos indica el estado del módulo.

A continuación, en la Figura 9 se muestra la asignación de pines del módulo HC-06. Se puede observar que la asignación de pines es análoga al módulo HC-05 salvo los pines de los extremos. De manera análoga a la semejanza a la asignación de pines, las especificaciones técnicas del dispositivo son iguales a las del módulo explicado anteriormente.

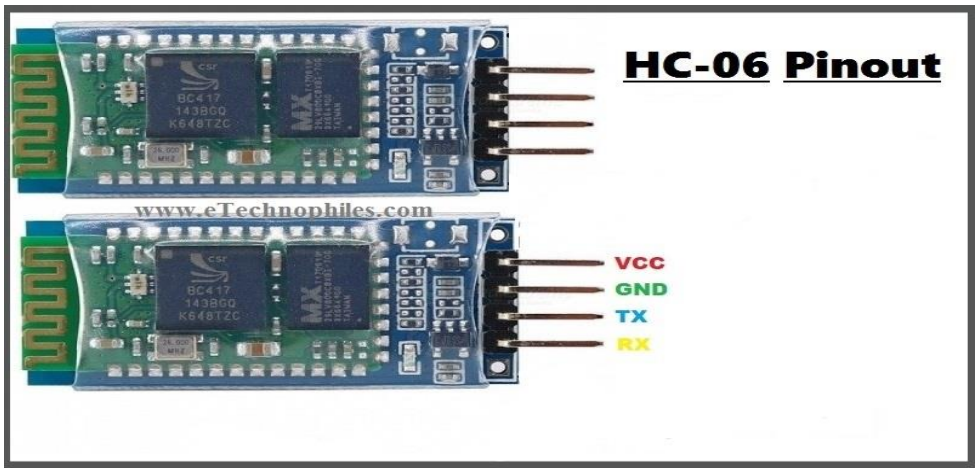

Figura 9: Asignación de pines del módulo HC 06

Estos módulos son los que en la actualidad tienen más stock en el mercado gracias a su facilidad de conexión, debido tanto a su clara asignación de pines como a que ya vienen estos soldados. Otra de su principal característica que viene derivada de su popularidad es que muchas placas de desarrollo ya vienen diseñadas para introducir directamente estos módulos, y de esta manera eliminar la dificultad y los diferentes problemas que puede desarrollar las conexiones mediante cables o soldadura. Debido a todas estas razones, este es el módulo que se utiliza para desarrollar este proyecto.

4.3 Protocolo de comunicación

En este apartado se analizarán los principales protocolos de envíos de datos en serie para la comunicación. En la comunicación serial se utiliza un solo cable para enviar los datos y otro para recibirlos, aunque puede ser utilizando el mismo cable en las dos direcciones, por lo tanto, es necesario utilizar el protocolo que se ajuste mejor a la aplicación.

4.3.1 I2C

I2C o *Inter-Integrated Circuit* es el protocolo serie más utilizado ya que es el único que se ha diseñado con el objetivo de conectar más de un dispositivo. En cuanto a las conexiones hardware, se puede observar en la Figura 10, que únicamente se necesitan dos cables, uno de ellos llamado SCL o *Serial Clock* para los pulsos de reloj y el otro llamado SDA o *Serial Data* para transmitir y recibir la información.

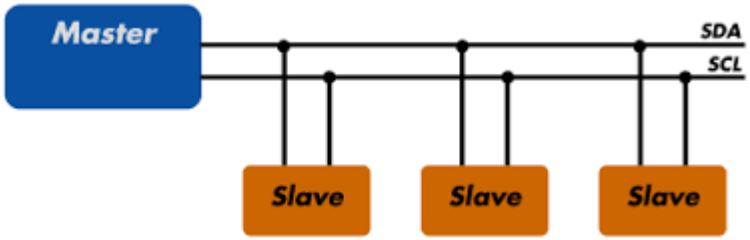

Figura 10: Conexión hardware I2C

En cuanto a la implementación software, el protocolo I2C se puede gestionar mediante librerías para facilitar su uso e implementación con sensores y periféricos. Como por ejemplo la librería *LiquidCrystal_I2C.h [1]* para Arduino que permite gestionar de forma sencilla una pantalla LCD conectada al bus I2C.

Este protocolo se define por permitir una cantidad de datos por envío elevada y a una velocidad media, 100kHz en modo estándar, debido a su funcionamiento Half-dúplex al disponer solo de un hilo por el cual transmitir y recibir la información. Un ejemplo de este envío se puede observar en la Figura 11. Esta comunicación se puede

realizar a una distancia máxima de 8 metros a un número de receptores de 127 para un direccionamiento de 7-bit y 1023 receptores si el direccionamiento es de 10-bit.

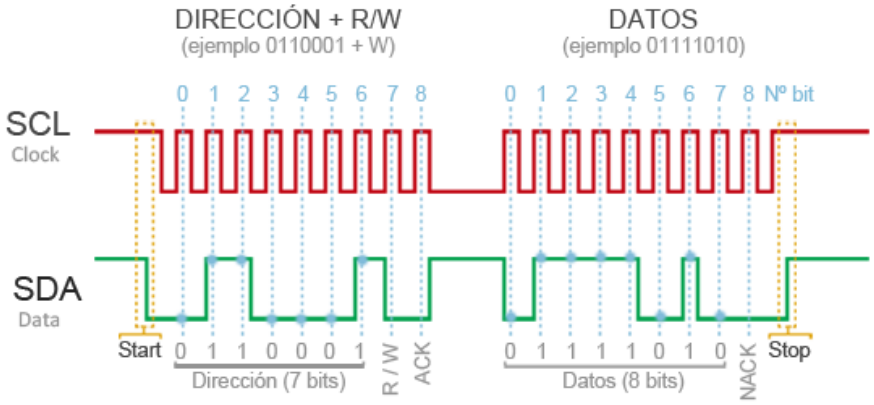

Figura 11: Envió de información utilizando el protocolo I2C

4.3.2 SPI

SPI o *Serial Peripheral Interface* es el protocolo serial con la velocidad más alta. Su implementación hardware se puede observar en la Figura 12 y consta de 4 pines, SCLK (*Serial Clock*) que se utiliza para los pulsos de reloj, MISO (*Master Input Slave Output*) para el envío de la información desde el esclavo al maestro, MOSI (*Master Output Slave Input*), para transmitir la información desde el maestro hasta el esclavo y SS (*Slave Select*) para, en el caso de tener más de un esclavo, establecer la comunicación con el esclavo correspondiente.

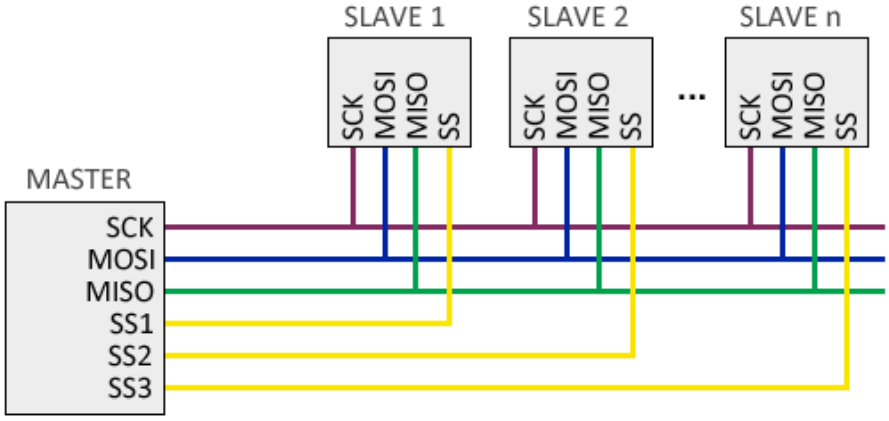

Figura 12: Conexión hardware SPI

De manera análoga al protocolo I2C, la implementación es sencilla en software. Esta se realiza mediante librerías ya implementadas por otros usuarios de este protocolo, como pueden ser *SPI.h [2]* en Arduino y *HAL_SPI [3]* en STM32

Este protocolo se define por la gran cantidad de datos que puede enviar controlados por una señal de reloj, un ejemplo de envío de datos se puede observar en

la Figura 13. Además, al utiliza el tipo de comunicación Full-dúplex gracias a los dos pines que dispone para la comunicación (MISO y MOSI) permite una velocidad de comunicación alta, de hasta 60Mbps a un número máximo de tres receptores.

Figura 13: Envío de información usando el protocolo SP

4.3.3 UART

UART o *Universal Asynchronous Receiver-Transmitter* es el protocolo con menor dificultad de implementación. Se puede observar en la Figura 14 que su conexión hardware consta de 2 pines, uno de ellos es el encargado de enviar la información (Tx) y otro el encargado de recibirla (Rx). La conexión se ha de realizar cruzada, haciendo coincidir el pin de transmisión de uno de los dispositivos con el pin de recepción del otro y viceversa.

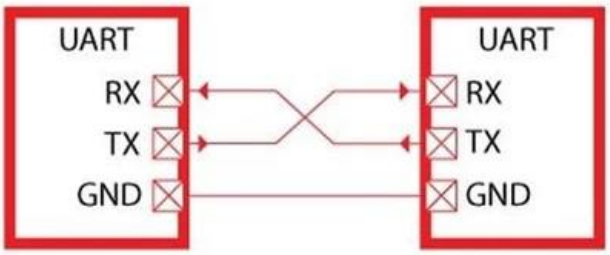

Figura 14: Conexión hardware UART

En cuanto a su implementación software, debido a lo extendido que se encuentra su uso, se pueden encontrar diferentes bibliotecas que permiten trabajar con ella y por tanto facilita su programación. Algunos ejemplos de estas bibliotecas son *ARM UART LIBRARY [4]* y *microC PRO UART LIBRARY [5].*

En cuanto a las principales características de este protocolo, en primer lugar, se encuentra la cantidad de datos que se pueden enviar, siendo 8 bits por vez como se puede observar en la Figura 15. A continuación, la velocidad de transmisión puede ser de hasta 1Mbps pudiendo ser Half-dúplex o Full-dúplex. El modo Half-dúplex es un modo de transmisión parcial en el que no se puede enviar y recibir información a la vez, mientras que el modo Full-dúplex permite una comunicación bidireccional constante. Por último, en la configuración Full-dúplex, el número de receptores que puede tener un maestro es limitada, normalmente siendo la relación 1 a 1,es decir, un esclavo por cada maestro.

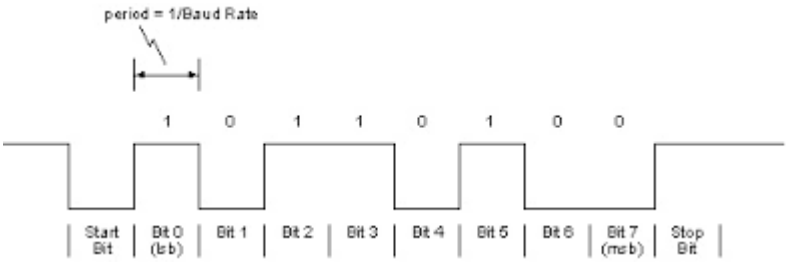

Figura 15: Envío de información usando el protocolo UART

Una vez analizados los diferentes protocolos de comunicación, se opta por hacer uso del UART. La razón de esta decisión es debido a la facilidad de su implementación a nivel de software y a que el dispositivo de comunicación inalámbrica seleccionado en el apartado anterior dispone de una interfaz de conexiones particular para utilizarlo.

4.4 Kit de desarrollo del mando

Una vez definido cómo, con qué dispositivo y con qué protocolo se va a realizar la comunicación, en este apartado se estudian las alternativas del mando. Este ha de

ser intuitivo y sus componentes han de tener las características necesarias para que se puedan implementar los elementos definidos en los apartados anteriores.

4.4.1 GAMEPI ATMEGA 32U4 DIY

Este kit de desarrollo es uno de los más solicitados cuando el desarrollador está comenzando. En la Figura 16 se puede observar el resultado final de este una vez montado.

Figura 16: GAMEPI ATMEGA 32U4 DIY

Como se puede observar en la Figura 16 el mando está formado por tres niveles y diferentes componentes. A continuación, se hará una descripción y análisis de estos.

En la Figura 17 se muestra el nivel inferior, en este nivel se encuentra la electrónica del mando. El componente de esta capa al que hay que prestar atención es el microcontrolador, que, como se indica en la Figura 17, es un ATMEGA32U4. Este microcontrolador se alimenta a 2.7V, tiene una interfaz de entrada/salida de 26 pines, una memoria Flash de 32KB, una memoria RAM de 2.5KB, una memoria EEPROM de 1KB, una velocidad de 16 MHz y admite conexiones I2C, SPI, UART y USB. Además, también se aprecia que en esta primera capa es donde va la alimentación, en este caso, mediante pilas AAA o equivalente.

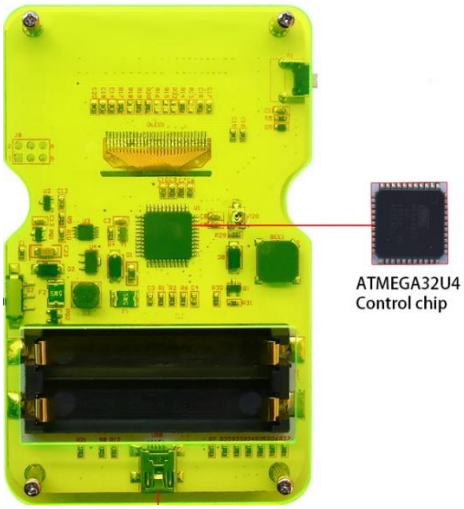

Figura 17: Capa inferior

A continuación, en la Figura 18, se muestra el nivel intermedio, en esta capa, se encuentran los periféricos con los que interactúa el usuario. En este caso, se distinguen dos componentes claros, el primero de ellos son los botones, que le proporcionan al usuario la capacidad de introducir una señal digital en el microcontrolador y una pantalla que permite al microcontrolador mostrarle al usuario la información.

Figura 18: Capa intermedia

Por último, la capa superior es la que permite el ensamblaje final. Esta capa únicamente aporta los huecos para colocar los embellecedores de los botones y el espacio para poder observar la pantalla, al acoplarla a las dos capas inferiores se obtiene el resultado presentado inicialmente en la Figura 16.

Como resumen de este mando, su microprocesador tiene todas las características que se han ido requiriendo de él en los apartados anteriores, además dispone de una interfaz intuitiva y que mediante la pantalla permite una buena comunicación con el usuario. Sin embargo, se ha de rechazar esta propuesta ya que al

plantear el proyecto siendo uno de los objetivos el movimiento final de un robot se echa en falta un periférico que permita comandar el movimiento mediante una señal analógica y no digital como el GAMEPI ATMEGA 32U4 propone.

4.4.2 Arduino Esplora

En la Figura 19 se muestra el Arduino Esplora, el nuevo kit de Arduino con forma de mando, basado en la famosa placa de Leonardo.

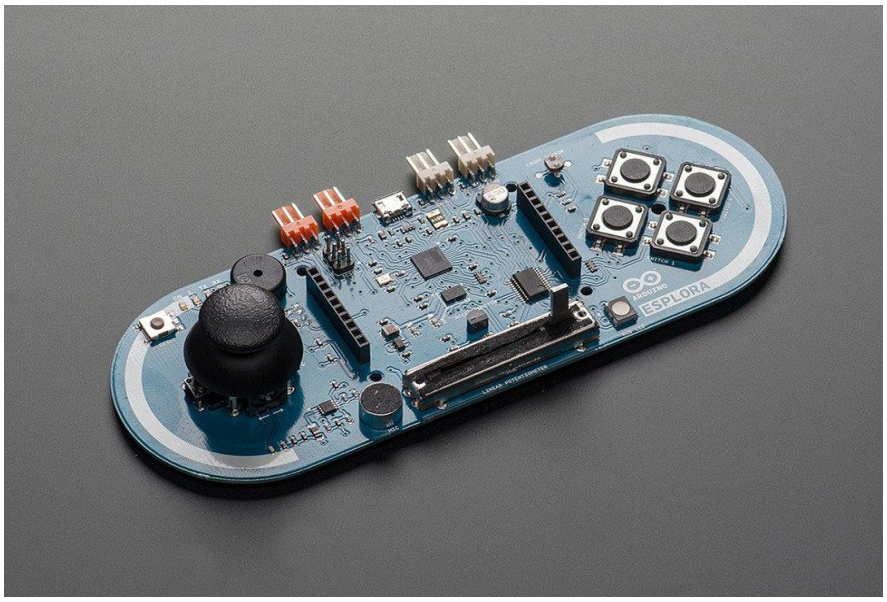

Figura 19: Arduino Esplora

Como se puede observar en la Figura 19, este kit dispone de un gran abanico de sensores y actuadores. En la Figura 20, se muestra el esquemático con los componentes más importantes referenciados.

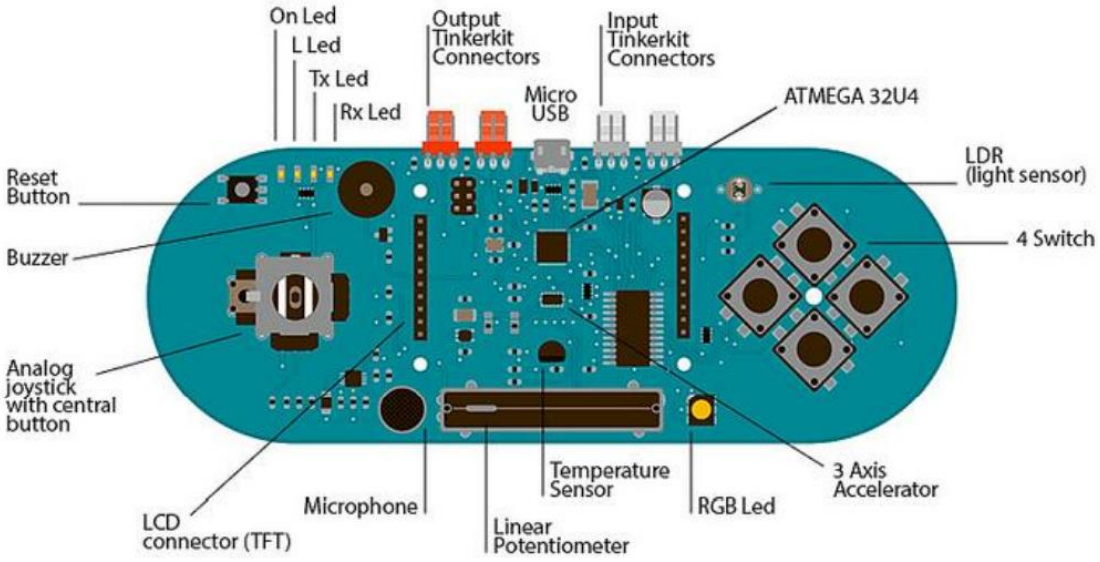

Figura 20: Esquemático de Arduino Esplora

El Arduino Esplora está comandado por el ATMEGA 32U4 que ya se ha introducido en el kit anterior. Además de este microcontrolador, también dispone de numerosos periféricos que son de interés. En primer lugar, se observan cuatro interruptores que proporcionan al usuario la capacidad de introducir señales digitales. Por otro lado, en la parte superior izquierda de la Figura 20, se observan cuatro indicadores LEDs, vitales para comprobar el correcto funcionamiento. Además, la placa ya está predispuesta a conectarle una pantalla LCD en el centro, ya que presenta las entradas para esta. Por último, el joystick permite al usuario introducir una señal analógica que puede aplicarse a la dirección del movimiento del robot.

En conclusión, tanto el microcontrolador ATMEGA 32UA como los diferentes periféricos que componen el Arduino Esplora encajan a la perfección con el objetivo del proyecto. Sin embargo, se descarta esta opción ya que se considera que no dispone de suficientes entradas disponibles ya predefinidas para la implementación en el futuro de nuevos componentes o mejoras.

4.4.3 BOOSTXL-EDUMKII y MSP-EXP432P401R

Esta opción, que se puede observar en la Figura 21, nace de la unión de las placas BOOSTXL-EDUMKII y MSP-EXP432P401 de Texas Instruments.

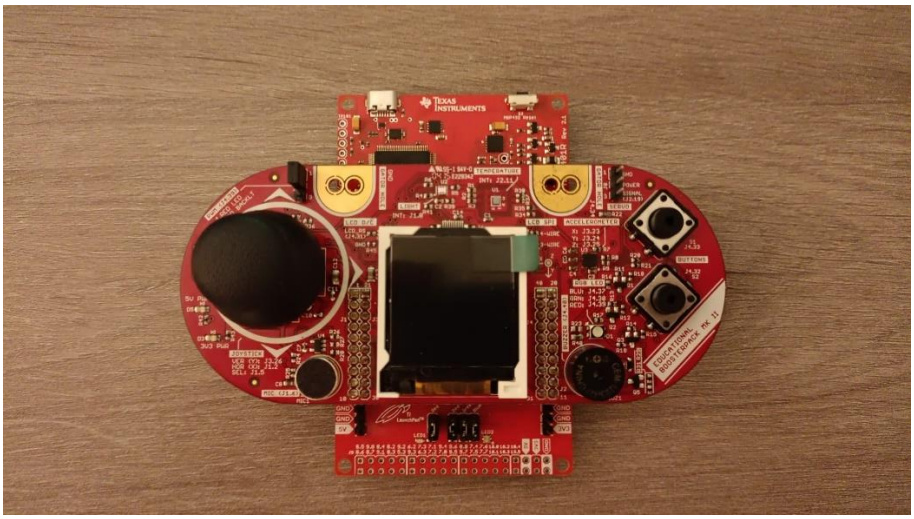

Figura 21: BOOSTCL-EDUMKII y MSP-EXP432P401R

Como se ha indicado anteriormente, este kit está formado por dos partes, en primer lugar, en la Figura 22 podemos observar la placa MSP-EXP432P401R y en la Figura 23 la descripción de sus componentes.

Figura 22: MSP-EXP432P401

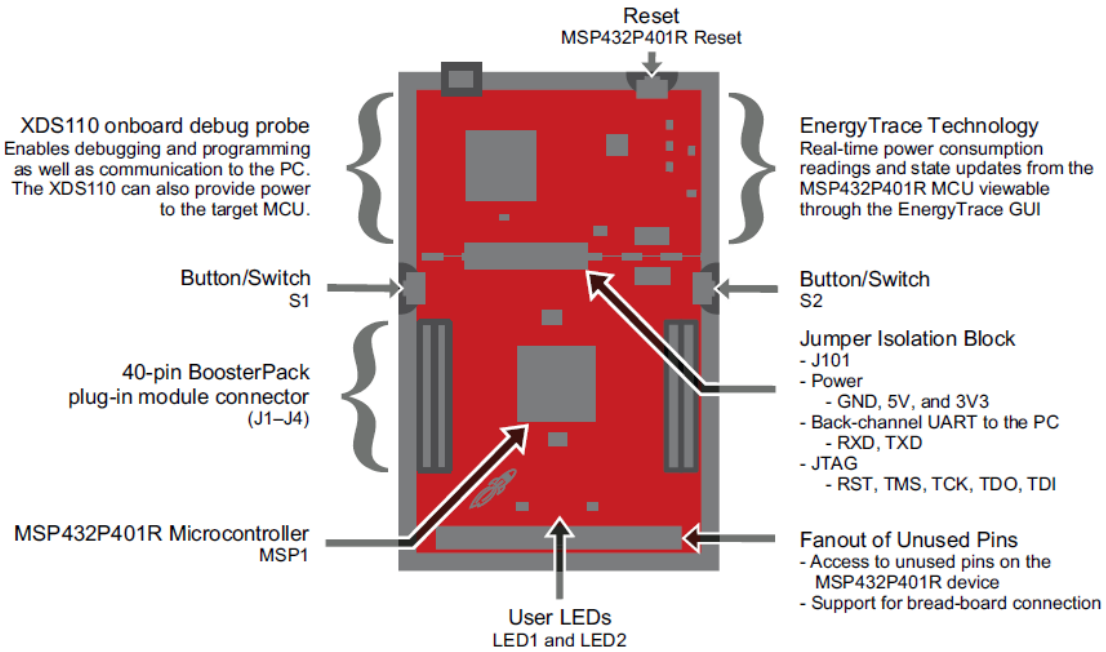

Figura 23: Descripción componentes MSP-EXP432P401

Como se puede observar en la Figura 23, la placa MSP432P401 utiliza un microcontrolador de baja consumo ARM Cortex-M4F MSP432P401R. Este trabaja a una frecuencia máxima de 48-MHz, poseyendo una capacidad de memoria flash de 256KB, 64KB de memoria SRAM y 32KB de ROM. Además, es capaz de comunicarse mediante el canal serie utilizando I2C, SPI, UART e IrDA. Por último, también dispone de numerosas salidas tanto digitales como analógicas. Por otro lado, también viene ya implementada en la placa la distribución de pines para la posterior conexión con la placa

BOOSTXL-EDUMKII y además de un botón de *reset*, dos entradas digitales y dos LEDs de usuario.

En la Figura 24 se muestra la parte superior de la placa BOOSTXL-EDUMKII. En la parte superior izquierda dispone de un *joystick*, que sirve para comandar la dirección del robot, en la parte derecha dos entradas digitales en modo de botones y en el centro una pantalla LCD que funciona como *display* para proporcionar información al usuario y pines de entrada y salida libres para la futura incorporación de nuevas funcionalidades. Esta placa, en su parte inferior dispone de las entradas para conectarla de manera directa a la placa MSP-432P401.

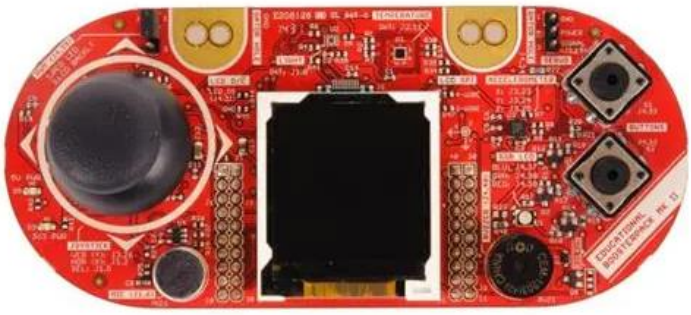

Figura 24: BOOSTXL-EDUMKII

En resumen, la unión de ambas placas forma un componente final que cumple con todas las necesidades para este proyecto. En cuanto a nivel de software, el microcontrolador ARM Cortex-M4F MSP432P401R cumple con la capacidad de comunicarse en el protocolo elegido anteriormente y permite la conexión con el dispositivo bluetooth a través de los pines de la BOOSTXL-EDUMKII. A nivel de hardware, esta opción proporciona tanto información al usuario a través de la pantalla, como un control analógico de la posición mediante el joystick además de disponer de dos entradas digitales para el usuario y numerosos pines libres para posteriores mejoras. Debido a esto, se selecciona este mando para el proyecto.

4.5 Kit de desarrollo robo

El último elemento del sistema que queda por definir es el robot. A lo largo de este apartado se analizarán diferentes opciones y se elegirá la opción que encaje mejor con el resto de los elementos definidos.

4.5.1 Alpha Bot 2

El Alpha Bot 2 es el robot más avanzado que ha desarrollado Seeed Studio dentro de su gama de educación, en la Figura 25 podemos ver su montaje final.

Figura 25: Alpha Bot 2

El material utilizado para el chasis del robot es aluminio. En el montaje de la estructura se puede ver que tanto la plancha superior como la inferior disponen de componentes eléctricos, sensores y actuadores, los más relevantes son descritos a continuación:

- Dos motores de corriente continua N20 que se muestran en la Figura 26. Estos funcionan a 6 V con un consumo de corriente medio de 100 mA, y 800mA de pico, alcanzando una velocidad máxima de 1000 RPM.

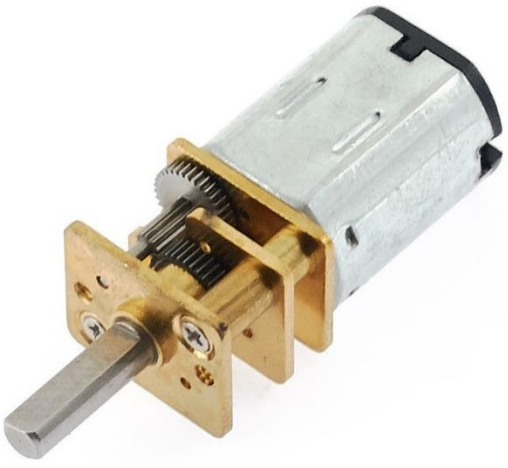

Figura 26: Motor DC N20

- Controladora de motores TB6612FNG que se muestra en la Figura 27. Esta controladora de motores funciona con una tensión de alimentación entre 2.7 y 5.5 V y puede alimentar a motores cuya alimentación sea entre 5 y 15 V, Por otro lado, permite la conexión de dos motores cuyo consumo de corriente sea menor o igual al consumo medio de esta, que es de 1.2 A.

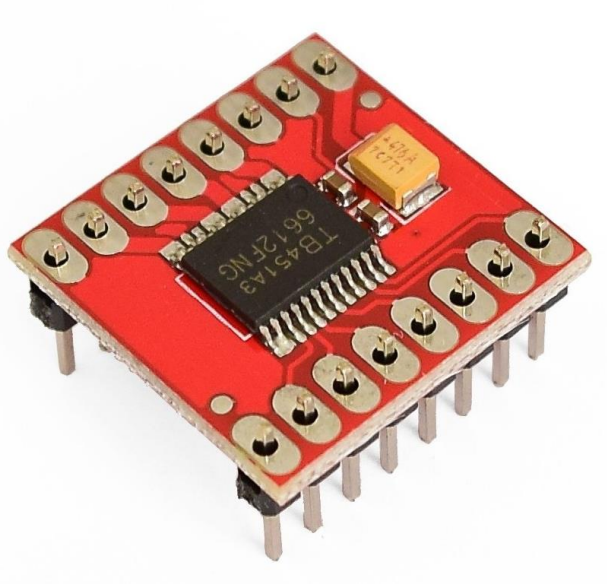

Figura 27: Controladora TB6612FG

- Sensor de ultrasonido HC-SR04. Este sensor que se enseña en la Figura 28 es el típico en los sistemas de Arduino gracias a su sencilla interfaz de cuatro hilos. (Vcc, trigger, echo, GND), Funciona alimentado a 5 V, con un consumo medio de 1.5mA y en un rango de 2 a 400 cm.

Figura 28: Sensor de ultrasonido HC-SR04

- En la Figura 29 se muestra la placa Uno Plus, que es la encargada de controlar al Alpha Bot 2. Esta placa cuenta con un microcontrolador ATMEGA328P-AU que es el encargado de ejecutar el código para hacer funcionar el robot. Este microcontrolador se ha de alimentar entre 1.8 y 5.5

V con un consumo de 0.3 mA. Posee capacidad de 32 KB de memoria FLASH, 2kB de memoria RAM y una máxima frecuencia de funcionamiento de 20MHz. Por otro lado, su interfaz de entrada-salida cuenta con 23 pines y 8 entradas que funcionan como convertidores analógico-digital. Además, admite comunicación utilizando I2C, SPI o UART/USART.

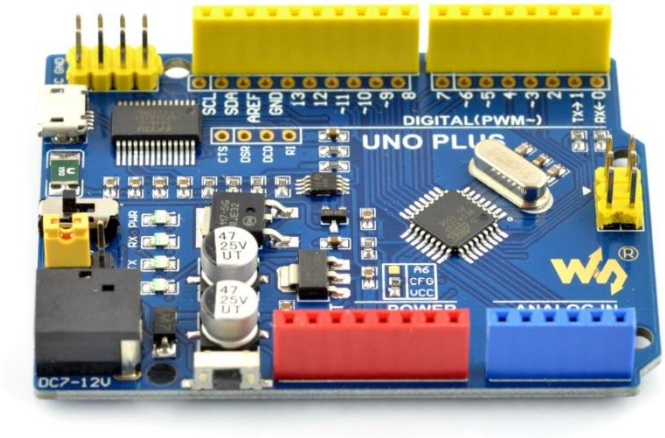

Figura 29: Placa Uno Plus

Como se ha mencionado en la descripción de los diferentes componentes principales, este kit de robot ya dispone en el conjunto de los principales componentes para esta aplicación, además de muchos adicionales que podrían incluirse en un futuro. Sin embargo, se ha de rechazar esta propuesta de robot debido a que este kit está preparado para utilizar un módulo bluetooth de la familia XBee y el módulo seleccionado anteriormente no pertenece a esta.

4.5.2 Robo Pro

En la Figura 30 se muestra el robot Robo Pro desarrollado por Ossep.

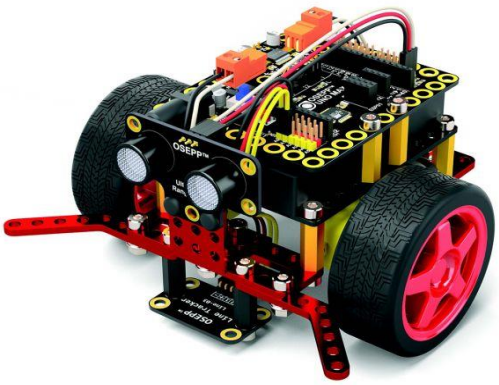

Figura 30: Robo Pro

Como se puede observar en la Figura 30, el robot está montado utilizando aluminio. En este montaje se puede observar que en la estructura se encuentran los diferentes componentes, los más significativos son descritos a continuación:

- Dos motores TTMT-01 que se muestran en la Figura 31. Estos funcionan entre 3 y 6 V en corriente continua. Su principal uso es en robots que son comandados con Arduino, tienen una velocidad máxima nominal de 200 RPM con su alimentación máxima y 90 RPM con la mínima.

Figura 31: Motor TTMT-01

- Controladora de motores TBMTD-01. En la Figura 32 se muestra que esta controladora dispone de dos entradas para los dos motores del robot, cada robot se conecta a esta con dos cables, uno para la dirección y otro para el control mediante PWM. Por último, necesita una alimentación entre 6 y 12 V y proporciona a cada motor un corriente constante de 1.2 A.

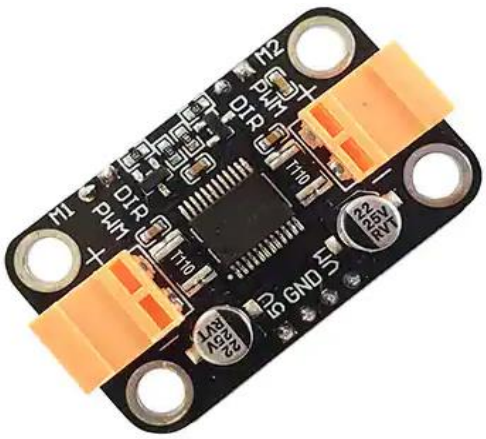

Figura 32:Controladora TBMTD-01

- Sensor de ultrasonido Ultra-01. Este sensor que se observa en la Figura 33, ha sido desarrollado por Osepp Electronics para trabajar con una alimentación de 5 V, con un consumo medio de 15 mA y para un rango de distancia de 3 cm a 1 m.

Figura 33: Sensor de ultrasonido Ultra-01

- Placa uno Max. Esta placa que se muestra en la Figura 34, es la encargada mediante el microcontrolador ATmega328P de gestionar todas las entradas y salidas que se ofrecen en la parte superior de la placa. Este microcontrolador trabaja con una tensión de alimentación entre 1.8 y 5.5 V con una frecuencia máxima de funcionamiento de 20MHz. Por otro lado, dispone de una capacidad de 32 KB en memoria flash, 2 KB en memoria RAM y 2 KB de EEPROM. Por último, no solo dispone de 23 canales de entrada y salida, sino que también dispone de 10 canales que funcionan como convertidores analógico-digital y, además utiliza los protocolos UART, TWI y SPI. Por último, esta placa lleva ya incorporado las entradas para conectar los diferentes elementos del kit y elementos estándar como módulos bluetooth, además de dejar muchos pines libres para el futuro.

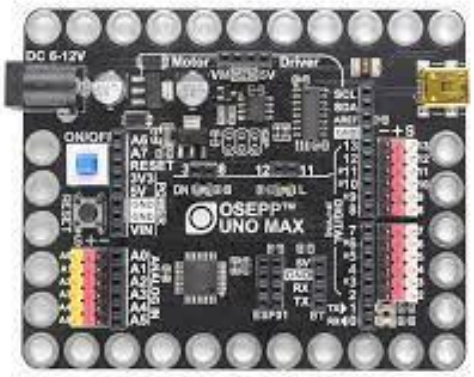

Figura 34: Placa Uno Max

Como se ha indicado en la explicación de componentes, este kit de robot ya lleva implementado todos los componentes necesarios para esta aplicación. Tanto los componentes hardware como los de software del kit cumplen todos los requisitos para ser elegido como el kit a utilizar. Además, la placa deja muchos pines libres tanto de entrada como de salida para el desarrollo futuro de mejoras en el robot. Debido a estas características y a lo ya explicado, se utiliza este kit para el desarrollo final.

5. Descripción de la solución adoptada

En este apartado se describe el proceso completo del desarrollo del proyecto, explicando el procedimiento paso a paso y el razonamiento de las diferentes decisiones que se han tomado.

El desarrollo se ha dividido en dos procesos principales, en primer lugar, se expone la implementación del mando, y a continuación el del robot.

Ambas implementaciones se descomponen en dos partes. Primeramente, se desarrolla lo relacionado con el hardware, y a continuación, se detalla la implementación a nivel de software.

5.5 Desarrollo mando

La implementación del mando en el proyecto se realiza en dos procesos claramente diferenciados. Por un lado, se encuentra la implementación hardware del sistema y por otro la implementación en software del algoritmo. El procedimiento de esta última parte se encuentra dividido en dos partes, en primer lugar, el planteamiento del algoritmo a implementar y, a continuación, la implementación de este en el lenguaje de programación C y en el entorno de programación Code Composer Studio ofrecido gratuitamente por Texas Instruments.

5.5.1 Implementación Hardware

En primer lugar, se han de identificar los componentes hardware y realizar una pequeña descripción de estos para conocer el material con el que se trabaja. El mando consta de tres componentes principales, la tarjeta MSP-EXP432P401R [6] explicada en el apartado 4.4.3 junto a otro de los componentes principales, la tarjeta de expansión BOOSTXL-EDUMKII [7] . El último de los componentes que forman el mando es el módulo de bluetooth HC-05 introducido en el apartado 4.2.3.

El primer paso para conseguir la implementación final del mando es juntar la tarjeta MSP-EXP432P401R con la tarjeta de expansión BOOSTXL-EDUMKII. Estas dos placas ya se encuentran diseñadas para un encaje directo de la una con la otra. Se ha de introducir los conectores hembras que se encuentran en la parte inferior de la tarjeta BOOSTXL-EDUMKII que se observan en la Figura 35 con los conectores machos situados en la parte superior de la tarjeta MSP-EXO432P401 que se muestran en la Figura 36.

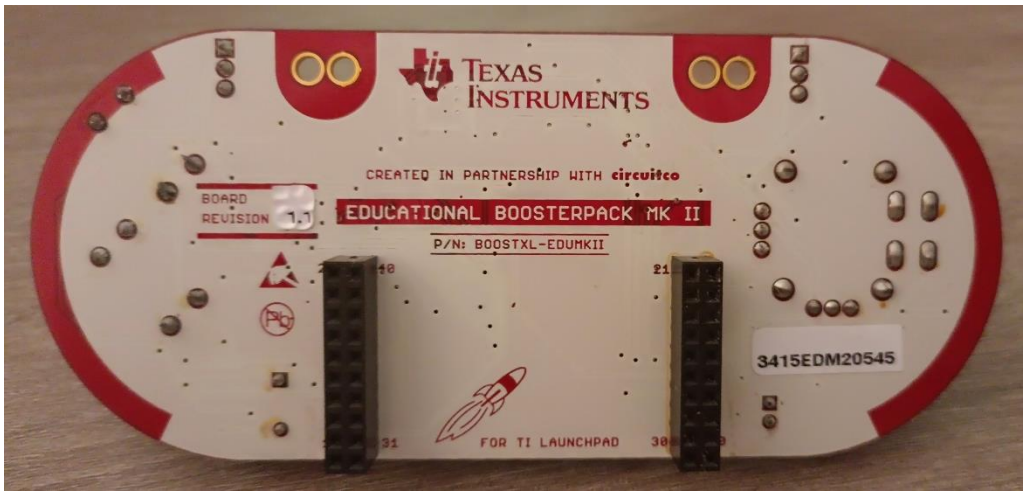

Figura 35: Parte trasera BOOSTXL-EDUMKII.

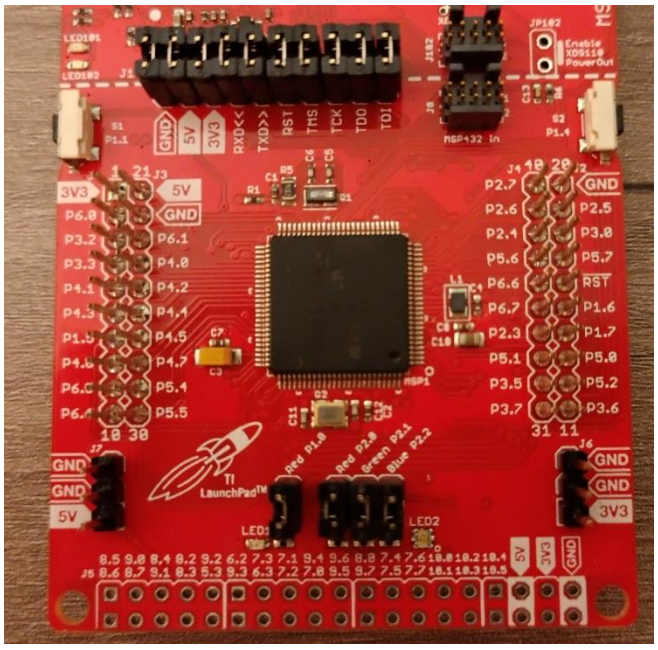

Figura 36: Pines de conexión de MSP-EXP432P401R.

A continuación, es necesario conocer qué tipo de señal van a introducir o requerir los diferentes periféricos y cuál va a ser su función. De todos los implementados en la tarjeta BOOSTXL-EDUMKII se ha decidido utilizar los siguientes:

- Pantalla LCD. Esta salida permitirá al microcontrolador mostrarle al usuario las diferentes opciones dentro de cada menú. La comunicación se realiza utilizando el protocolo SPI gestionándose mediante la biblioteca Crystalfontz128x128_ST7735.h. La conexión de la pantalla se realiza de manera interna a través de las conexiones de la tarjeta, pero se puede

monitorizar estas conexiones a través de los pines que se muestran en la Tabla 1.

BoosterPack Header Connection	Pin Function
J1.7	LCD SPI clock
J2.13	LCD SPI chip select
J2.15	LCD SPI MOSI
J4.31	LCD reset pin
J4.39	LCD backlight

Tabla 1: Asignación de pines pantalla LCD

- Ambos botones como entrada. Estos periféricos cada vez que sean pulsados introducirán en el microcontrolador una señal de nivel alto como la que se observa en la Figura 37. Estas entradas permitirán al usuario interactuar con los diferentes niveles del menú. Estas señales de entrada pueden monitorizarse utilizando los pines que se muestran en la Tabla 2.

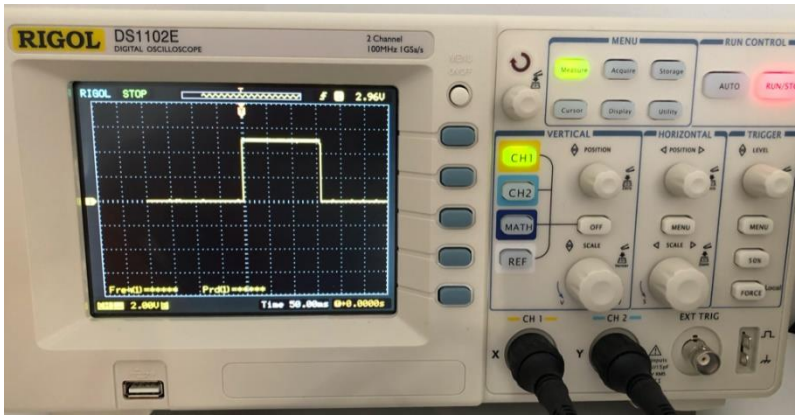

Figura 37: Señal de entrada que proporcionan los botones

BoosterPack Header Connection	Pin Function
J4.32	S2 button
J4.33	S1 button

Tabla 2: Asignación de pines de los botones

- El joystick. Este periférico introduce una señal analógica para la posición tanto en el eje X como en el eje Y cuyo valor oscila entre 0 y 16383. Al ser una señal analógica, se procesa en el microcontrolador a través de un convertidor analógico-digital (ADC). Como se observa en la Figura 38, la

posición X e Y tienen como referencia positiva (H+ y V+) 3.3V y como referencia negativa GND (H- y V-). El joystick funciona como un potenciómetro y el valor que lee el microcontrolador para la posición X e Y (H, V) oscila entre las dos referencias. El ADC realiza un proceso de muestreo de esta entrada y la transforma a un valor conformado por unos y ceros. El microcontrolador ARM Cortex-M4F posee un ADC de 14 bits, por lo tanto, convierte el valor analógico con un rango de 0 a 3.3 V a un valor digital de 0 a 16383. A estas señales se puede acceder utilizando los pines que se muestran en la Tabla 3.

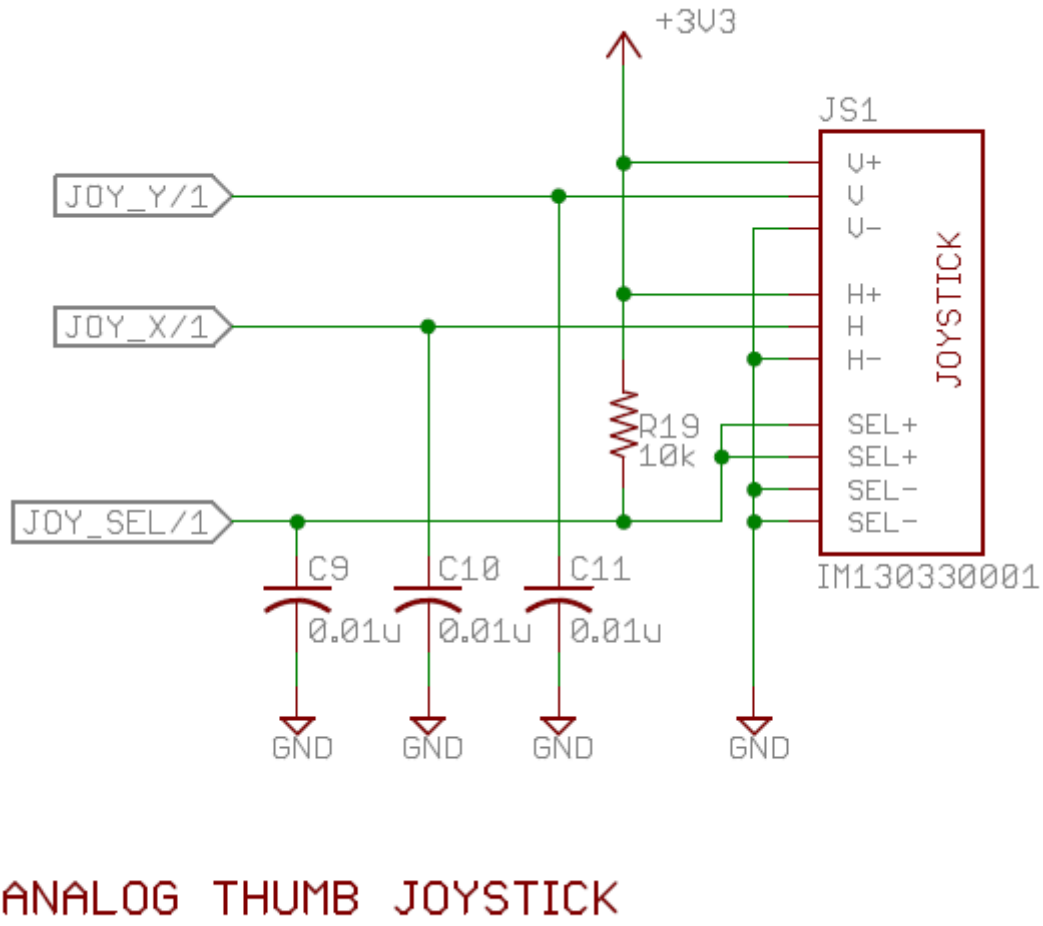

Figura 38: Conexión Interna Joystick

BoosterPack Header Connection	Pin Function
J1.2	Horizontal X-axis
J3.26	Vertical Y-axis

Tabla 3: Asignación de pines del joystick

Por último, falta conectar el módulo de bluetooth al mando. Para ello, siguiendo la asignación de pines que se mostró en la Figura 8, es necesario conectar el pin de Vcc del bluetooth al pin de 5V de placa, los pines Rx y Tx del módulo con el opuesto de la placa y conectar ambos GNDs. En la Tabla 4 se hace un resumen de la conexión a realizar entre la placa y el módulo bluetooth. En la Figura 39 se muestra el conjunto de mando y bluetooth.

BoosterPack Header Connection	Pin Function	Pin Bluetooth
J3.21	Vcc	Vcc
J3.22	Gnd	Gnd
J1.3	Rx	Tx
J1.4	Tx	Rx

Tabla 4: Asignación de pines para la conexión con el módulo bluetooth

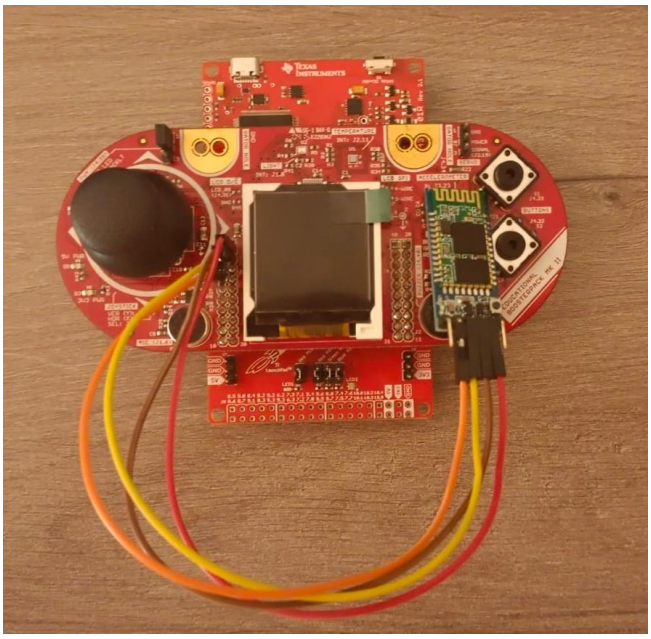

Figura 39:BOOSTCL-EDUMKII, MSP-EXP432P401R y módulo bluetooth

5.5.2 Implementación Software

Una vez conectados los diferentes elementos que conforman el hardware del mando, en este apartado se desarrolla la implementación del código. En primer lugar, se explica de manera conceptual y esquematizada mediante diagramas de flujo el algoritmo y a continuación la implementación de este concepto en el entorno de programación Code Composer Studio desarrollado por Texas Instrument y utilizando las librerías propias de los periféricos de la tarjeta que se encuentran en [8].

En primer lugar, el código del mando se divide en tres pantallas principales, como se muestra en la Figura 40, pantalla 1, a la que se le llama menú 1, y de igual manera,

a las pantallas 2 y 3 se les llama menú 2 y 3. El movimiento entre menús se realiza utilizando los dos botones físicos del mando, siendo el superior derecho el denominado Botón A y el inferior izquierdo como Botón B.

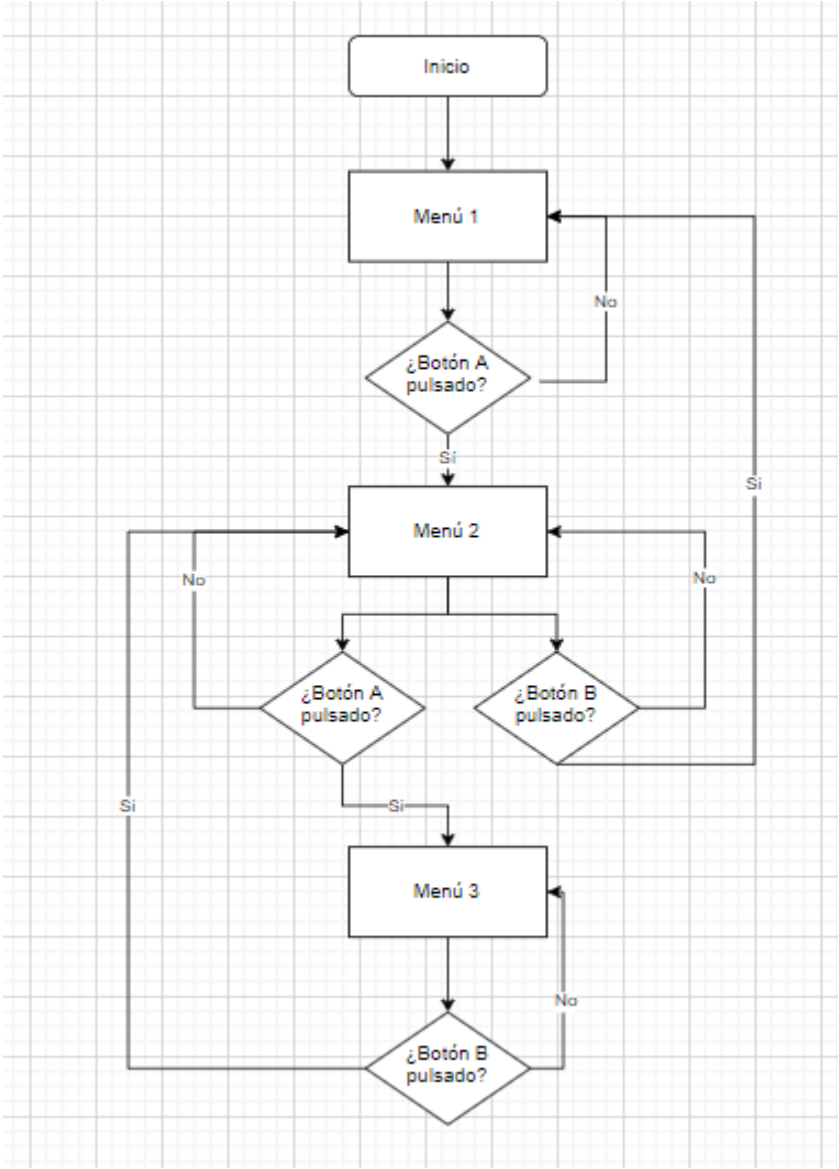

Figura 40: Diagrama de funcionamiento general.

El funcionamiento de los menús es el siguiente:

- El primer menú consiste en una espera simple, el mando queda en reposo hasta que el usuario pulsa el botón A para avanzar al menú 2. El menú 1 se puede observar en la Figura 41.

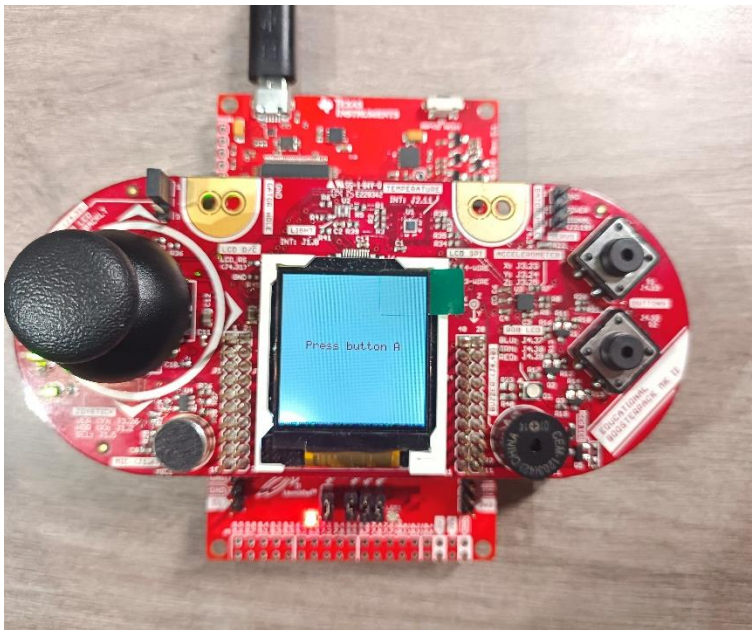

Figura 41: Menú 1

- El menú numero 2 es donde el usuario decide el modo de funcionamiento que se observan en la Figura 42. El desplazamiento entre estos modos de funcionamiento que se encuentran dentro del menú 2 se realiza con el joystick, para seleccionar cualquiera de ellos y continuar al menú 3 se ha de presionar el botón A y para volver al menú 1, se ha pulsar el botón B.

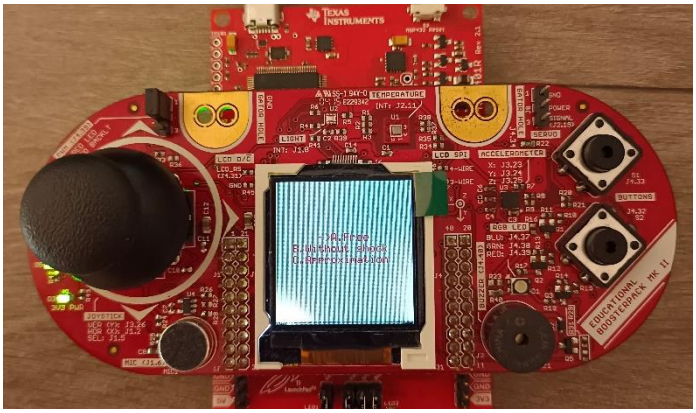

Figura 42: Menú 2

- En el último menú 3, como se observa en la Figura 43 , solamente se indica el modo de funcionamiento elegido anteriormente. Mientras que se esté en este menú, el mando se encuentra enviando constantemente información de la posición del joystick al coche para controlar su posición. Para salir de este modo se ha pulsar el botón B y se accede al menú 2.

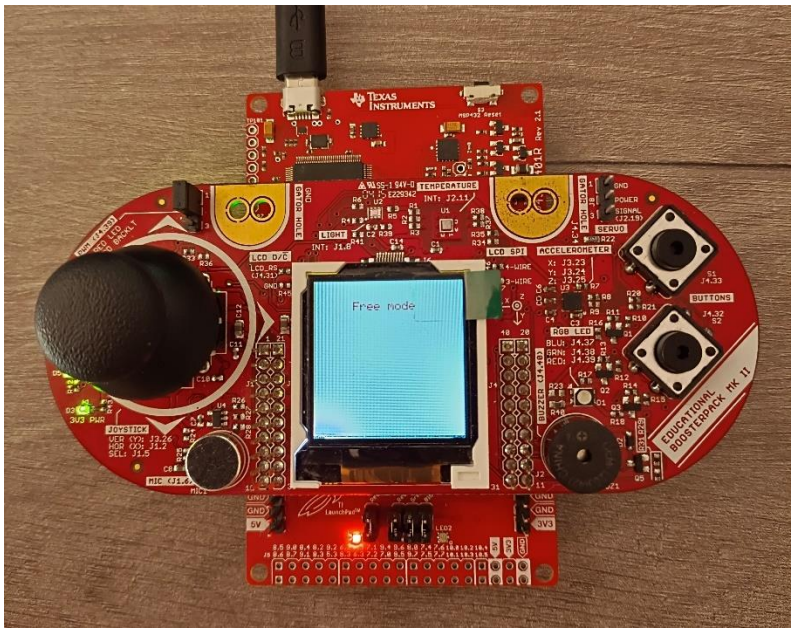

Figura 43: Menú 3

Ya explicada la idea de desarrollo, se procede con la implementación de esta. Para ello, se abre la aplicación del entorno de programación Code Composer Studio y se crea un nuevo proyecto.

Antes de comenzar con la explicación del código, en la Tabla 5 se muestran las variables globales que son utilizadas a lo largo del código.

Nombre	Tipo	Descripción
Pos	uint8_t	Posición relativa del *joystick* en el menú 2.
y	uint8_t	Posición del texto en la pantalla LCD
menu	uint8_t	Guarda el menú en el que se encuentra el mando
read1	uint16_t	Usada para gestionar la interrupción del botón A
read2	uint16_t	Usada para gestionar la interrupción del botón B
aux	uint8_t	Almacena la cuenta del manejador *SysTick*
remainder	uint32_t	Almacena el resto de una división
quotient	uint8_t	Almacena el cociente de una división
change	uint8_t	Utilizada como semáforo para las interrupciones
string	char	Utilizada para escribir en la pantalla LCD

Tabla 5: Variables globales código mando

En la Figura 44 se muestra la función *main()* y en esta se puede ver cómo, primeramente, se inician y configuran los diferentes elementos que se utilizan a lo largo de la ejecución. A continuación, se entra en un bucle infinito donde se va llamando a cada uno de los menús en función de la variable *menu* que comanda una estructura switch-case.

```
int main(void)
 {
    WDT_A_holdTimer();              /* Stop watchdog timer */
    initsystem();
    initdysplay();
    initadc();
    inituart();
    initpin();
    Interrupt_enableMaster();

    ledoff(Port1,Pin0);
    read1=1;
    read2=1;

    while(1)
    {
        switch (menu)
        {
        case 1:
            delay(1000);
            menu1();
            break;
        case 2:
            delay(1000);
            menu2();
            break;
        case 3:
            delay(1000);
            menu3();
            break;
        }
    }
 }
```

Figura 44: Función main()

Como se observa en la Figura 44, el bucle principal del código llama a la función menu1, esta se muestra en la Figura 45.

```
void menu1(void)
{
    y=60;

    writedisplay("Press button A");
    change = 1;
    while(read1 == 1  )
    {
        ledon(Port1,Pin0);
    }

    clearlcd();

    menu=2;
    read1=1;
}
```

Figura 45: Función menu1()

Como ya se ha indicado, esta primera pantalla es solo un menú de espera. Cuando se entra en esta función, se escribe un mensaje en la pantalla a través de la función *writedisplay();* Y se mantendrá esta pantalla hasta que la variable *read1* que es

la asociada a la interrupción que genera el botón A cambie de valor. Una vez que esta variable de control cambia, se limpia la pantalla del mensaje anterior, se actualiza la variable *menu* y se vuelve a otorgar a la variable *read1* el valor de 1 para el siguiente menú.

Para que se puedan mostrar mensajes en el display, en primer lugar, se ha iniciar la comunicación utilizando la librería *Crystalfontz128x128_ST7735.h*. Esta inicialización se consigue utilizando las funcione que se muestran en la Figura 46, que corresponde a la función *initdisplay(),* que es llamada al inicio de la función *main()* para mantener esta configuración a lo largo de toda la ejecución. Dentro de esta inicialización de la pantalla, se configura el tamaño del display, se asigna un color al fondo y otro a las letras, además de indicar la orientación para que aparezca el contenido.

```
void initdysplay(void)
{
    /* Initializes display */
    Crystalfontz128x128_Init();

    /* Set default screen orientation */
    Crystalfontz128x128_SetOrientation(LCD_ORIENTATION_UP);

    /* Initializes graphics context */
    Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128, &g_sCrystalfontz128x128_funcs);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_WHITE);
    GrContextFontSet(&g_sContext, &g_sFontFixed6x8);
    Graphics_clearDisplay(&g_sContext);
}
```

Figura 46: Función initdisplay()

Una vez inicializada, en la Figura 47 se muestra la función *writedisplay()* que es la encargada de mostrar los mensajes en la pantalla ya inicializada. A esta función se le envía una cadena de caracteres y utilizando una de las funciones de la librería *Crystalfontz128x128_ST7735.h* se envía este texto a la pantalla para que se muestre. Con la variable *y*, introducida en la Tabla 5, se decide qué altura de la pantalla aparece el texto en el *display*. La parte superior corresponde con el valor 0, y cada vez que el valor incrementa, el texto va desplazándose hacia abajo en la pantalla.

```
void writedisplay(char word[20])
{
    //char string[15];
    sprintf(string, "%s", word);
    Graphics_drawStringCentered(&g_sContext,
                                (int8_t *)string,
                                AUTO_STRING_LENGTH,
                                64,
                                y,
                                OPAQUE_TEXT);
    y=y+10;
}
```

Figura 47: Función writedisplay()

Para poder navegar entre los diferentes menús, se han de actualizar las variables *read1* y *read2*. Ambas variables se actualizan en el manejador de la interrupción que se genera en su pin al detectar este el cambio de nivel lógico que genera el botón asociado. En la Tabla 6 se observa el pin y puerto asociado a cada uno de los botones.

Botón	Puerto	Pin
A	5	1
B	3	5

Tabla 6: Pin y puerto de los botones

Antes de poder acceder al manejador de las interrupciones, es necesario realizar la configuración de los pines para activar esta interrupción, esta acción se muestra en la Figura 48 que corresponde a una parte de la función *initpin()* que es llamada en la función *main()*.

```
GPIO_setAsOutputPin(Port1, Pin0); // Pin1 output
GPIO_setAsInputPinWithPullUpResistor(Port3, Pin5); //Button 1 --> input || 1 Not pressed --> 0 presses
GPIO_setAsInputPinWithPullUpResistor(Port5, Pin1); //Button 2 --> input || 1 Not pressed --> 0 pressed

GPIO_enableInterrupt(Port3, Pin5); //Init Interrupt
GPIO_enableInterrupt(Port5, Pin1); //Init Interrupt
GPIO_clearInterruptFlag(Port3, Pin5); //Clear flag
GPIO_clearInterruptFlag(Port5, Pin1); // Clear flag
GPIO_interruptEdgeSelect(Port3, Pin5, GPIO_LOW_TO_HIGH_TRANSITION); // 0 --> 1
GPIO_interruptEdgeSelect(Port5, Pin1, GPIO_LOW_TO_HIGH_TRANSITION); // 0 --> 1

GPIO_registerInterrupt(Port3,PORT3_IRQHandler);
GPIO_registerInterrupt(Port5,PORT5_IRQHandler);
```

Figura 48: Configuración de los pines de los Botones A y B

Con esta configuración de cada uno de los botones, cada vez que se pulse uno de ellos se accederá de manera automática a las funciones *PORT3_IRQHandler()* y *PORT5_IRQHandler(),* a la primera de estas se accederá cuando se pulse el botón B y a la otra cuando se pulse el botón A. En la Figura 49 se puede observar que la

implementación de ambos manejadores de interrupción es análoga, siendo el único cambio la variable a modificar y el pin y puerto con el que se trabaja. Al entrar en la interrupción, se desactiva esta y se limpia el flanco del puerto para que no se vuelva a activar la interrupción mientras se está en el manejador. A continuación, se comprueba el estado de la variable *change*, esta variable es la que comprueba si se ha terminado la actualización anterior de las variables *read1* y *read2* y antes de salir de la función se vuelve a habilitar la interrupción para futuros usos. Esta comprobación mediante la variable *change* se realiza para evitar que un rebote o una mala pulsación del botón cree dos interrupciones y por tanto cambie dos o más veces seguidas la variable *read1* o *read2*.

```
void PORT5_IRQHandler(void) //Button A
{
    GPIO_disableInterrupt(Port5,Pin1);
    GPIO_clearInterruptFlag(Port5,Pin1);
    if(change == 1)
    {
        read1 = 0;
        change = 0;
    }
    GPIO_enableInterrupt(Port5,Pin1);
}

void PORT3_IRQHandler(void) //Button B
{
    GPIO_disableInterrupt(Port3,Pin5);
    GPIO_clearInterruptFlag(Port3,Pin5);
    if(change==1)
    {
        read2 = 0;
        change = 0;
    }
    GPIO_enableInterrupt(Port3,Pin5);
}
```

Figura 49: Implementación de los manejadores de interrupción

Tras pulsar el botón A en el menú 1, se pasa al menú 2, se puede ver la implementación de este menú en la Figura 50.

```
void menu2(void)
{
    ledoff(Port1,Pin0);
    while(read1 == 1 && read2==1)
    {
        senddata(255);
        senddata(255);
        y=50;

        switch (pos){
        case 1:
            writedisplay("->A.Free");
            writedisplay("B.Without shock ");
            writedisplay("C.Approximation ");
            break;
        case 2:
            writedisplay("A.Free ");
            writedisplay("->B.Without shock ");
            writedisplay("C.Approximation  ");
            break;
        case 3:
            writedisplay("A.Free  ");
            writedisplay("B.Without shock  ");
            writedisplay("->C.Approximation ");
            break;
        }

        pos=readpos();
        delay(150);
        change = 1;
    }
    clearlcd();
    if(read1==0)
    {menu=3;read1=1;}
    else
    {menu=1;read2=1;}
}
```

Figura 50: función menu2()

En este menú es donde se puede elegir el modo de funcionamiento. Para ello, hay que desplazarse entre los diferentes modos de funcionamiento utilizando el joystick. Para leer la posición del joystick se utiliza la función *readpos().* Esta función, que se observa en la Figura 51, es la encargada de leer el valor analógico mediante la función *readjoystick()* y dependiendo del valor que esta devuelva se actualiza la variable *pos*.

```
void  readpos(void)
{
    uint16_t value[2];
    readjoystick(value);

    if(value[1]>10000)
    {pos--;}
    if(value[1]<6000)
    {pos++;}
    if(pos<1)
    {pos=1;}
    if(pos>3)
    {pos=3;}

}
```

Figura 51: función readpos()

Una vez que se tiene elegido el modo de funcionamiento, se ha de pulsar el botón A para entrar a este y en el caso de que se quiera volver al menú 1, se ha de pulsar el botón B para que cambie la variable *read1* o *read2* y se cambie al menú correspondiente.

La primera línea que se ejecuta tras entrar al bucle *while* es la función *senddata(),* esta función es la encargada de enviar la información al módulo bluetooth para que se comunique con el coche. Esta función *senddata(),* que se muestra en la Figura 52, utiliza la función *UART_transmitData()* que viene ya definida en la biblioteca *uart.h* para enviar mensajes utilizando este protocolo.

```
void senddata(uint_fast8_t data)
{
    UART_transmitData(EUSCI_A2_BASE,data);
}
```

Figura 52: Función senddata()

Para que ese mensaje se envíe de manera óptima se ha de realizar una configuración previa. Esta operación previa se realiza en la función *inituart()*, donde, como se muestra en la Figura 53, en primer lugar, se define la función que realizan los pines conectados al módulo bluetooth, a continuación, se realiza la configuración de la uart, asignando los valores oportunos para que la velocidad de transmisión de datos sea de 9600 baudios.

```
void inituart(void)
{

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P3, GPIO_PIN2, GPIO_PRIMARY_MODULE_FUNCTION);
    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P3, GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);

    const eUSCI_UART_ConfigV1 uartConfig =
    {
     EUSCI_A_UART_CLOCKSOURCE_SMCLK, // SMCLK Clock Source
     312, // BRDIV = 78
     8, // UCxBRF = 2
     0, // UCxBRS = 0
     EUSCI_A_UART_NO_PARITY, // No Parity
     EUSCI_A_UART_LSB_FIRST, // LSB First
     EUSCI_A_UART_ONE_STOP_BIT, // One stop bit
     EUSCI_A_UART_MODE, // UART mode
     EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION, // Oversampling
     EUSCI_A_UART_8_BIT_LEN // DataLegth
    };

    /* Configuring UART Module */
    MAP_UART_initModule(EUSCI_A2_BASE, &uartConfig);
    /* Enable UART module */
    MAP_UART_enableModule(EUSCI_A2_BASE);

}
```

Figura 53: función inituart()

Estos parámetros introducidos para obtener el valor correspondiente de velocidad de comunicación se obtienen de la página web que proporciona Texas

Instrument y se muestra en la Figura 54. Aunque se muestre que estos cálculos son para la UART del microcontrolador MSP430 se puede utilizar igualmente para configurar el microcontrolador MSP432, pues ambas UART son idénticas.

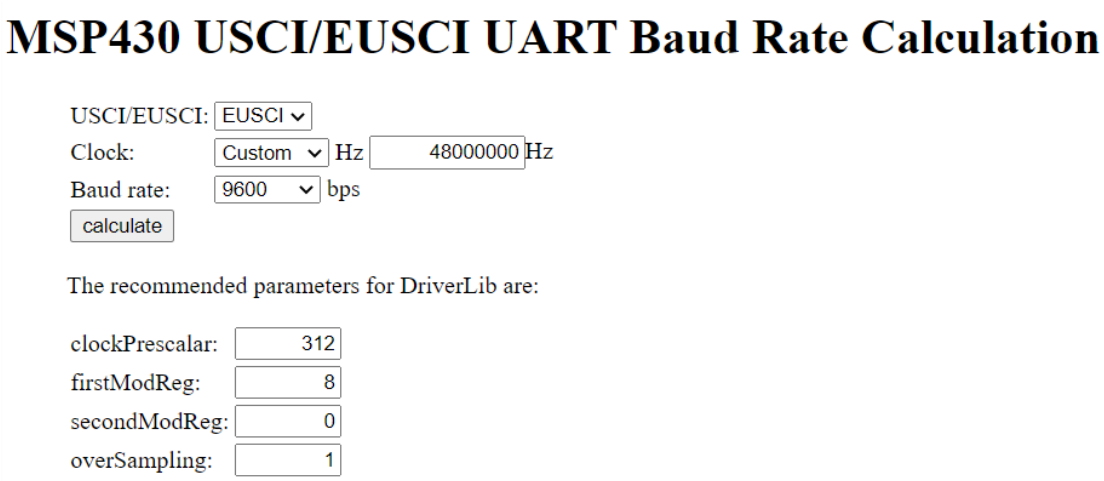

Figura 54: Parámetros para obtener la velocidad de comunicación

Por último, queda el último menú, el número 3. Se puede observar la implementación de este en la Figura 55.

```
void menu3(void)
{
    while (read2==1)
    {
        change = 1;
        y=80;
        ledon(Port1,Pin0);
        switch (pos)
        {
        case 1:
            delay(2000000);
            senddata(1);
            writedisplay("Free mode ");
            move();
            break;
        case 2:
            delay(2000000);
            senddata(2);
            writedisplay("Without shock ");
            move();
            break;
        case 3:
            delay(2000000);
            senddata(3);
            writedisplay("Approximation");
            move();
            break;
        }
        senddata(255);
        senddata(255);
    }

    menu=2;
    read2=1;

    clearlcd();
}
```

Figura 55: función menu3()

Como se muestra en la figura anterior, la implementación de este menú sigue la misma estructura que las dos anteriores. La principal diferencia de este menú reside en que es cuando se realiza la comunicación con el coche, es decir, únicamente enviaremos datos cuando nos encontremos en este modo. Se han implementado los

tres modos de funcionamientos de manera análoga. En primer lugar, una vez en el menú anterior se ha seleccionado el modo de funcionamiento se realiza una espera mediante la función *delay()*, a continuación, se envía un único dato, este envío se utiliza para que el receptor entre en el mismo modo de funcionamiento que el emisor. Una vez se ha realizado esta primera comunicación, se muestra por pantalla el modo de funcionamiento que se ha elegido y se entra en la función *move()*, que es la encargada de enviar constantemente los datos de posición y que se muestra en la Figura 56.

```
void move(void)
{
    static uint16_t results[2];
    uint_fast8_t k1;
    uint_fast8_t k2;
    uint_fast8_t mapresults[2];
    while(read2==1)
    {
        delay(100);
        readjoystick(results);
        mapread(results,mapresults);
        k1=mapresults[0];
        k2=mapresults[1];
        senddata(k1);
        delay(100);
        senddata(k2);
    }

}
```

Figura 56: Función move()

Cada vez que se ejecuta la función *move(),* se leen la posición tanto horizontal como vertical en la que se encuentre el joystick mediante la función *readjoystick(),* este valor obtenido, que oscila entre 0 y (2^{14} - 1), se transforma en un valor entre 0 y 255 utilizando la función *mapread()* para enviarse a través de la función *senddata()*.

Finalmente, tras implementar todo el código, disponible en el Anexo 1 Código mando de este documento, se ha de obtener en la pantalla LCD los menús que se han mostrado anteriormente.

5.6 Desarrollo robot

La implementación del robot en el proyecto se realiza en dos procesos claramente diferenciados de la misma manera que se hizo con el mando. Por un lado, se encuentra la implementación de los diferentes elementos que componen el hardware y por otro la implementación en software del algoritmo. El procedimiento de esta última parte se encuentra dividido en dos partes, en primer lugar, el planteamiento del algoritmo a implementar y, a continuación, la implementación de este en el lenguaje de programación C y en el entorno de programación de Arduino.

5.6.1 Implementación Hardware

La conexión del módulo de bluetooth se realiza utilizando las conexiones predefinidas que se encuentran ya en la placa del robot que se puede observar en la Figura 57

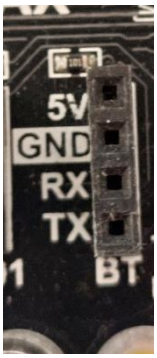

Figura 57: Conector rápido de módulo bluetooth en el robot

En la Figura 58 se muestra el montaje final del robot.

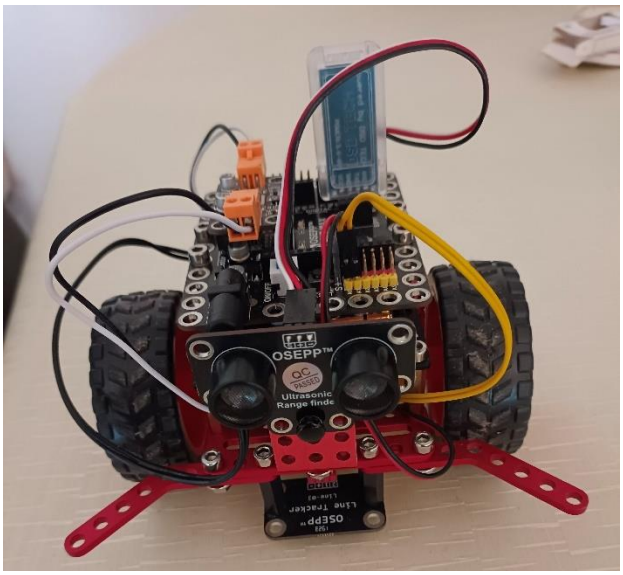

Figura 58: Montaje final del robot

5.6.2 Implementación Software

Al igual que se ha explicado en el apartado 5.1.2, este apartado se divide en dos partes, la primera que consiste en una explicación conceptual y la segunda donde se desarrolla el código implementado en el entorno de programación de Arduino.

El concepto que se va a implementar en el microcontrolador que comanda el coche se puede observar en la Figura 59. El programa consta de tres modos de funcionamiento, el primero de ellos, denominado modo *free,* permitirá al robot moverse libremente bajo la supervisión del usuario que lo controla a través del mando. A continuación, se encuentra el modo *without shock*, este modo permite un movimiento libre del robot, pero cuando el sensor de ultrasonido detecte que hay un obstáculo a una distancia inferior de 15 cm, automáticamente se cancelará el movimiento hacia delante, para así evitar colisiones. Por último, el modo *approximation*, sigue con la misma idea del anterior, pero, la distancia a la cual se cancela el movimiento hacia delante cada vez es menor. Es decir, en primer lugar, la distancia es de 15 cm, tras llegar por primera vez a esta distancia, automáticamente la distancia pasa a reducir su valor en 3 cm. Esto se realiza de manera continúa hasta que se llegue a una distancia de 3 cm, una vez en esta distancia ya no se permite el movimiento hacia delante. Con este modo se permite al usuario acercarse de manera segura sin choque a cualquier objeto.

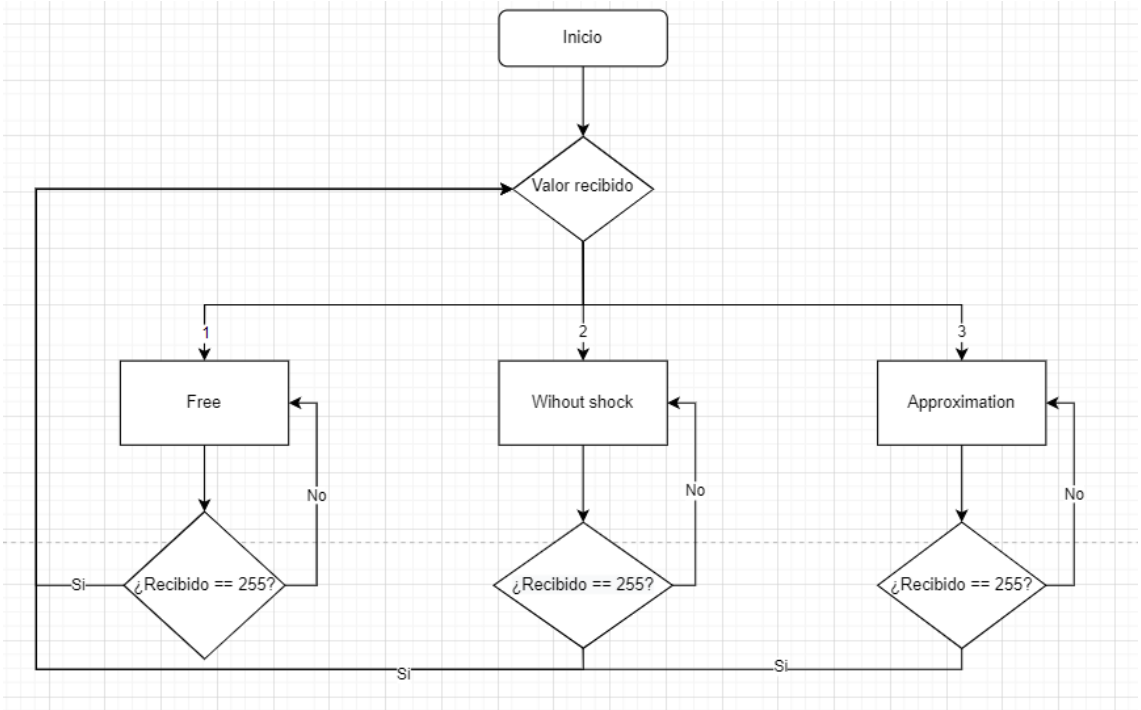

Figura 59: Diagrama de funcionamiento general

Como se puede observar en la Figura 59 el valor que decide el modo de funcionamiento oscila entre el 1 y 3, mientras que se sale de estos modos cuando el valor recibido sea de 255.

Ya explicada la idea de desarrollo, se procede con la implementación de esta. Para ello, se abre la aplicación del entorno de programación de Arduino y se crea un nuevo archivo.

Antes de comenzar con la explicación del código, en la Tabla 7 se muestran las variables globales que son utilizadas a lo largo del código.

Nombre	Tipo	Descripción
DISTANCE	float	Constante de distancia de frenado
direct	uint8_t	Permite al robot ir hacia delante
stop	bool	Auxiliar de parada
sp	uint8_t	Almacena la velocidad del motor derecho
sp2	uint8_t	Almacena la velocidad del motor izquierdo
mode	uint8_t	Guarda el modo de funcionamiento
distance	float	Almacena la distancia leída por el sensor de ultrasonido
b	byte	Almacena la lectura del canal serie

Tabla 7: Variables globales código robot

Como ya se ha introducido en la Figura 59, para entrar en uno de los modos, se ha de recibir un valor entre 1 y 3 y dependiendo de este se entra en uno de los modos de funcionamiento. Por otro lado, para salir de estos modos se ha recibir el número 255.

Para que esta lectura se produzca, en la Figura 60 se muestra la función *readSerial()*. Está función es la encargada de leer los datos que son recibidos por el bluetooth y los almacena en una variable. Para que lo implementado en la función se ejecute de manera correcta, es necesario introducir la instrucción *Serial.begin(9600)* en la función *setup()*. Esta instrucción es la encargada de asignar la velocidad de comunicación al número que se le introduzca, en este caso 9600. Esta velocidad de comunicación, 9600 baudios, es la misma que se le ha asignado en la Figura 53 al mando para una correcta sincronización.

```
void readSerial(byte b[8])
{
  if (Serial.available() > 0)
  {
    Serial.readBytes(b, 8);
  }
}
```

Figura 60: función readSerial()

Otra de las características que ya se ha introducido es la lectura de la posición mediante un sensor de ultrasonido, para que esta lectura funcione de manera correcta se implementa la función *measuredistance()* que se muestra en la Figura 61.

```
float measuredistance(void)
{
  pinMode(US, OUTPUT);
  digitalWrite(US, LOW);
  delayMicroseconds(2);
  digitalWrite(US, HIGH);
  delayMicroseconds(5);
  digitalWrite(US, LOW);
  pinMode(US, INPUT);
  unsigned long tm = pulseIn(US, HIGH);
  float distance = tm / 58.0;
  delayMicroseconds(50);
  //Serial.println(distance);
  return distance;
}
```

Figura 61: función measuredistance()

Para el movimiento, es necesario inicializar ambas ruedas. El controlador de motores admite una onda PWM, por tanto, será necesario conectar este a los pines que ya vienen predispuestos para enviar esta señal. Con el objetivo de inicializar las ruedas, se ha creado un *struct* para asignar los 3 pines que corresponden a cada rueda. Con los pines ya asignados, se utiliza la función *initwheel*() se inicializan los valores para que cuando comience la ejecución las ruedas permanezcan paradas. Este proceso se puede observar en la Figura 62.

```
typedef struct
{
  uint8_t dir_pin;
  uint8_t pwm_pin;
  uint8_t forward_level;

} Wheel;

Wheel Left_Wheel = {12, 11,LOW};
Wheel Right_Wheel = {8, 3, HIGH};

void initwheel(const Wheel side)
{
  pinMode(side.dir_pin, OUTPUT);
  digitalWrite(side.pwm_pin, LOW);
}
```

Figura 62: Configuración de las ruedas

En la Figura 63 se muestra la estructura switch-case que se encuentra en el bucle *void loop()* que es el principal del programa. En este bucle se utiliza la función *readSerial()* para actualizar la variable *mode* que es la que gobierna la estructura switch-case. Dependiendo del valor que obtenga la variable *mode*, se selecciona el modo de funcionamiento.

```
switch (mode)
{
  case 1:
    readSerial(ref);
    modefree(ref[0], ref[1]);
    mode=0;
    forward(0,Right_Wheel);
    forward(0,Left_Wheel);
    break;
  case 2:
    readSerial(ref);
    modeWS(ref[0], ref[1]);
    mode=0;
    forward(0,Right_Wheel);
    forward(0,Left_Wheel);
    break;
  case 3:
    readSerial(ref);
    modeAprox(ref[0], ref[1]);
    mode=0;
    forward(0,Right_Wheel);
    forward(0,Left_Wheel);
    break;
  default:
    mode=0;
    break;
}
```

Figura 63: Estructura switch-case dentro del bucle principal

En esta figura se utiliza la función *forward()*, esta función que se muestra en la Figura 64 es la encargada de realizar el movimiento de los motores.

```
void forward(int pwm, const Wheel side)
{
  if (pwm < 0)
  {
    digitalWrite(side.dir_pin, !side.forward_level);
    pwm = -pwm;
  }
  else
  {
    digitalWrite(side.dir_pin, side.forward_level);
  }
  analogWrite(side.pwm_pin, pwm > 255 ? 255 : pwm);
}
```

Figura 64: función forward()

Por último, las funciones *modefree()*, *modeWS()* y *modeAprox()* son las encargadas de ejecutar su modo de funcionamiento correspondiente. Debido a la extensión de estas, no se incluyen las figuras completas durante la explicación, para su consulta se puede observar el Anexo 1.2 Código Robot.

En primer lugar, la función *modefree()* entra en un bucle *while* que se ejecuta mientras que los valores recibidos en las variables X e Y sean diferentes de 255. Durante la ejecución de este bucle, se lee constantemente el *buffer* del canal serie que contiene la información enviada por el mando y se almacena la primera posición del *buffer* en la variable Y mientras que la segunda posición se guarda en la variable X. Una vez obtenido las dos coordenadas del movimiento se llama a la función *forward()* que es la encargada de mandar la señal de movimiento a los motores.

A continuación, la función *modeWS()*, sigue el mismo procesamiento para el movimiento dentro de un bucle. La diferencia es que se le añade un semáforo, para que cuando la distancia sea menor a la predefinida no pueda ir hacia delante y obligue al usuario a retroceder. La implementación de esta función antichoque se observa en la Figura 65 . La variable *stop* se utiliza para indicar que el robot se encuentra detenido. Por otro lado, cuando la variable *direct* vale 0 permite el movimiento del hacia delante del robot y cuando se encuentra a 1 únicamente permite el movimiento hacia atrás del robot para retroceder y evitar el coche.

```
distance = measuredistance();
if (distance < DISTANCE )
{
  direct = 0;
  if (stop == false)
  {
      forward(0,Right_Wheel);
      forward(0,Left_Wheel);
      stop = true;
  }
}
else
{
  if (distance > DISTANCE)
  {
      direct = 1;
     stop=false;
  }
}
```

Figura 65: Estructura del semáforo

Por último, la función *modeAprox()* combina el movimiento de la función *free()*, con el semáforo antichoque de la función *modeWS()*. Además, en esta función se implementa, como ya se ha comentado antes, un acercamiento progresivo hacia un objeto. Para conseguir esto, se ha implementado la estructura de la Figura 66. En esta estructura, cada vez que se ejecute el bucle, con una pequeña espera para que el acercamiento sea suave, se va a ir decrementando el valor en el cual se hace la espera hasta llegar al valor límite donde se activa el semáforo explicado anteriormente, en ese momento el robot únicamente tendrá permitido el movimiento hacia atrás.

```
if (distance < dist  )
{
  forward(0,Right_Wheel);
      forward(0,Left_Wheel);
  delay(300);
  dist=dist-3;
}
 if (distance < 2.99)
{
  direct = 0;
   if (stop == false)
  {
      forward(0,Right_Wheel);
      forward(0,Left_Wheel);
      stop = true;
  }
}
else if (distance > 3.00)
{
    direct = 1;
    stop = false;
}
```

Figura 66: Estructura de aproximación y semáforo

Finalmente, se realiza la combinación de ambos códigos y se consigue el resultado final de controlar la dirección y el movimiento del robot mediante el mando.

5.7 Emparejamiento módulos bluetooth

Una vez realizados los dos códigos, el siguiente paso es vincular los módulos para que la comunicación funcione. Para que el emparejamiento sea exitoso es necesario sincronizar ambos módulos a la misma velocidad (9600 baudios).

Para poder realizar la configuración es necesario utilizar el IDE de Arduino y conectar los pines de comunicación y alimentación del módulo con una tarjeta de Arduino. Esta conexión de pines consiste en conectar Vcc del módulo con +3.3V o +5V de la placa, unir los pines GND y cruzar los pines de comunicación, es decir, Rx del módulo con Tx de la placa y Tx del módulo con Rx de la placa.

Una vez realizada la conexión y abierto el IDE de Arduino se abre el monitor serie que se puede encontrar en la pestaña de herramientas de la barra superior o utilizando el atajo Ctrl+Mayús+M.

Ya realizado estos pasos, debemos seleccionar Sin ajuste de línea y 9600 baudio en los desplegables que se encuentran en la esquina inferior derecha del monitor serie.

Con el monitor serie ya configurado, se ha enviar a los módulos los comandos AT necesarios para su configuración. En primer lugar, se envía el comando AT\r\n y el módulo devuelve un OK si todo es correcto. Seguidamente se envía el comando que indica si el módulo funciona como esclavo o maestro. Al módulo HC-05 el comando a enviar es AT+ROLE=1\r\n mientras que al módulo HC-06 el comando es AT+ROLE=0\r\n. Por último, se ha de seleccionar la velocidad de funcionamiento de estos, para ello se utiliza el comando AT+UART=9600\r\n.

Una vez configurados ambos módulos, se ejecuta el comando AT+ADDR=? para obtener la dirección física bluetooth de cada módulo. Una vez obtenida esta dirección, se utiliza el comando AT+BIND en el módulo opuesto para establecer la dirección física bluetooth a la cual cada módulo se conecta al encenderse, es decir, la dirección física del módulo HC-05 ha de introducirse en el módulo HC-06 y viceversa.

Con estos pasos realizados, se conectan ambos bluetooth a una fuente de alimentación y se espera hasta que la frecuencia de ambas luces disminuya.

6. Conclusiones.

6.1 Conclusiones sobre el trabajo realizado

A lo largo de este trabajo se han trabajado los conceptos básicos para desarrollar una comunicación inalámbrica entre microcontroladores a través de un mando con una interfaz de periféricos intuitiva y un robot que permita comprobar que está comunicación es efectiva. Partiendo de esta base, esta idea se puede implementar para el desarrollo de sistemas más complejos, como sensorización a distancia o envío de datos en remoto.

En el desarrollo del proyecto se han aplicado una gran parte de los conocimientos adquiridos en el grado de ingeniería, como puede ser la electrónica digital y analógica, la lectura de esquemáticos y la programación de microcontroladores. Además, en el proceso de realización se han desarrollado nuevas habilidades y conocimientos que se utilizarán en el futuro.

Aunque muchos de los aspectos del proyecto se pueden mejorar, debido a la falta de recursos y tiempo esto no ha sido posible. Sin embargo, se puede concluir que el objetivo que se planteó al inicio del trabajo ha sido alcanzado con éxito, ya que la comunicación entre mando y robot ha sido efectiva y se ha conseguido el movimiento a distancia del robot como se demuestra en los videos del siguiente enlace https://www.youtube.com/channel/UC45SotRiDMp2MptwezOj9ow.

6.2 Futuras Mejoras

Con el proyecto finalizado, se abre un abanico de posibilidades para que, basándose en este trabajo, se desarrollen una gran cantidad de productos. Mas allá de los objetivos que se establecieron en el inicio del proyecto, a continuación, se presentan una serie de mejoras e ideas para completarlo.

Uno de los aspectos que se puede mejoras es optimizar la interfaz del mando añadiendo diferentes periféricos. Esta mejora nos permitiría mejorar y perfeccionar las ordenes que recibe el robot y, por tanto, realizar una comunicación más efectiva.

Otro aspecto donde se puede mejorar es el método utilizado para dirigir el robot. Ya que se tuvieron en cuenta estas mejoras a la hora de seleccionar componentes, quedan suficientes entradas y salidas para implementar una cámara de visión artificial que permita mediante una aplicación y el mando enviar una posición en el espacio para que el robot se dirija hacia ella.

Aparte de introducir la cámara en el robot, también sería interesante incluir actuadores, como pinzas, para permitir al robot agarrar y desplazar objetos y así poder utilizar el control remoto para mover objetos a distancia y sin esfuerzo.

7. Bibliografía

[1] John, R. (17 de Julio de 2020). *Github*. Recuperado el 16 de Julio de 2022 Obtenido de https://github.com/johnrickman/LiquidCrystal_I2C/

[2] Arduino. (27 de Junio de 2022). *Arduino*. Recuperado el 17 de Julio de 2022, de https://www.arduino.cc/reference/en/language/functions/communication/spi/

[3 Magdy, K. (2 de Mayo de 2021). *DeepBlue*. Recuperado el 17 de Julio de 2022, de DeepBlue: https://deepbluembedded.com/stm32-spi-tutorial/]

[4] ARM. (s.f.). *ArmMBED*. Recuperado el 17 de Julio de 2022, de https://os.mbed.com/teams/Embedded-System/wiki/ARM-UART-Library

[5] mikroC. (s.f.). *mikroe*. Recuperado el 17 de Julio de 2022, de https://www.arduino.cc/reference/en/language/functions/communication/spi/

[6] Texas Instruments. (March de 2015). *User's Guide MSP432P401R SimpleLink™ Microcontroller LaunchPad™.* Obtenido de https://docs.rs-online.com/3934/A700000006811369.pdf

[7] Texas Instruments. (Agosto de 2015). *User's Guide BOOSTXL-EDUMKII Educational BoosterPack™ Plug-in.* Recuperado el 3 de Julio de 2022, de https://www.ti.com/lit/ug/slau599b/slau599b.pdf?ts=1662020548810

[8] Texas Instruments. (2016). *Texas Instruments MSP432 Driver Library.* Recuperado el 21 de Julio de 2022, de https://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP432_Driver_Library/3_21_00_05/exports/driverlib/msp432_driverlib_3_21_00_05/doc/MSP432P4xx/html/driverlib_html/index.html

Universitat Politècnica de Valéncia

Escuela Técnica Superior de Ingeniería del Diseño

Comunicación inalámbrica entre microcontroladores para controlar un robot de forma remota.

2. PLIEGO DE CONDICIONES

Autor: Montesinos Berenguer, Daniel

Tutor: Rodríguez Ballester, Francisco

Curso Académico: 2021/2022

Índice Pliego de Condiciones

1. Definición y alcance del pliego

El presente documento consiste en la programación de dos microcontroladores para su comunicación con el objetivo de controlar el movimiento de un robot mediante un mando de manera inalámbrica. El fondo del proyecto es de carácter educativo, con el objetivo de conocer algunas de las arquitecturas de microcontroladores que se encuentran en el mercado y aplicar de esta manera alguno de los conocimientos aprendido en el desarrollo del grado.

Los microcontroladores seleccionados para el proyecto se encuentran basados en las tecnologías desarrolladas por las familias ATmega328P en el robot y ARM Cortex-M4F en el mando. La familia ATmega328P se programa en la plataforma de Arduino mientras que para la familia ARMCortex-M4F se utiliza el IDE Code Composer Studio.

En cuanto al desarrollo e implementación del código, se ha de tener en cuenta tanto las características particulares de ambas placas de desarrollo como de los periféricos que se han planteado en este documento.

Conociendo la variedad que se encuentra en el mercado de los diferentes componentes que se han seleccionado, el desarrollo de este proyecto solo asegura el funcionamiento con los productos que se han descrito en la memoria de este proyecto.

El objeto de este pliego de condiciones es seleccionar y fijar los criterios y las características que se han de valorar a la hora de replicar o implementar este proyecto.

Para este proyecto se han de tener en cuenta principalmente elementos electrónicos. En algunos casos, estos componentes pueden ser sustituidos por elementos que tengas características similares siempre y cuando cumplan los requisitos mínimos que se explican en este documento.

2. Condiciones y normas de carácter general

2.1 Instalación

Previo a la realización del proyecto se debe comprobar que se dispone de todos y cada uno de los elementos que componen el hardware y se dispone de todas las aplicaciones software necesarias.

Para el correcto desarrollo del proyecto que se plantea se debe tener conocimientos de carácter técnico en programación y electrónica.

2.2 Seguridad

Ha de disponerse de certificados de cumplimiento de la normativa vigente en el momento del desarrollo del proyecto de cada uno de los componentes y herramientas que son utilizadas en el transcurso del proyecto.

La alimentación externa que utilizan tanto el robot como el mando debe contar con la correcta protección para evitar posibles accidentes. Además, se debe comprobar que estas funcionen correctamente y tengan un nivel de carga adecuado.

2.3 Utilización

Este proyecto tiene un objetivo educativo, cualquier uso que sea diferente al planteado puede ser peligroso. Las posibles modificaciones que se realicen sobre la idea planteada han de ser analizadas y estudiadas de manera minuciosa para evitar funcionamientos inesperados o inadecuados.

2.4 Mantenimiento

Antes de utilizar los elementos que definen el proyecto se han de realizar una serie de verificaciones para un correcto funcionamiento.

En primer lugar, se ha de inspeccionar que las conexiones son correctas y los cables se encuentran en un estado óptimo para el funcionamiento. Tras esto, se ha de comprobar que los diferentes periféricos se encuentren en buen estado y la señal que se obtiene de ellos es la esperada.

Se ha de comprobar que la batería que lleva el robot tiene suficiente energía para la correcta alimentación de los periféricos que este lleva incorporado.

Por último, es necesario mantener los elementos del sistema protegidos de largas exposiciones a la luz solar, altas temperaturas y humedad.

3. Condiciones particulares

3.1 Condiciones técnicas

Las características técnicas mínimas del microcontrolador han de ser las siguientes:

El microcontrolador del robot ha de tener las siguientes características:

- Familia ATmega329
- Alimentación entre 1.8 y 5.5 V
- Frecuencia mínima de funcionamiento 5MHz
- Un mínimo de 16 canales de entrada y salida
- Un mínimo de 6 canales ADC
- Un mínimo de 4 canales PWM
- Protocolo serie UART, I2C y SPI

El microcontrolador del mando ha de tener las siguientes características

- Familia ARM Cortex-M4F
- Alimentación entre 2.5 y 6.6 V
- Frecuencia mínima de funcionamiento 10 MHz
- Un mínimo de 6 canales ADC
- Un mínimo de 4 canales PWM
- Protocolo serie UART, I2C y SPI

El módulo bluetooth ha de tener las siguientes características

- Alimentación entre 1.5 y 5.5 V
- Interfaz directa para UART

Los motores ha de tener las siguientes características:

- Alimentación entre 2 y 6 V
- Velocidad nominal entre 80 y 250 RPM
- Motor DC

El driver ha de tener las siguientes características:

- Dos salidas
- Control por PWM
- Alimentación entre 6 y 12 V
- Corriente suministrada entre 1.8 y 3 A.

El sensor de ultrasonido ha de tener las siguientes características

- Alimentación entre 2.5 y 5.5 V
- Consumo medio de corriente entre 5 y 20 mA
- Rango de medida 3 cm a 500 cm

3.2 Condiciones legales

Este proyecto se publica bajo la licencia pública general (GNU GPLv3), por lo que se permite el uso, estudio y modificación de este proyecto siempre que se tenga en cuenta el nombre del desarrollador original del presente proyecto. Además, el autor del presente no se hace responsable de los posibles problemas causados por un uso diferente al cual se ha diseñado.

Universitat Politècnica de Valéncia

Escuela Técnica Superior de Ingeniería del Diseño

Comunicación inalámbrica entre microcontroladores para controlar un robot de forma remota.

3. PRESUPUESTO

Autor: Montesinos Berenguer, Daniel

Tutor: Rodríguez Ballester, Francisco

Curso Académico: 2021/2022

Índice Presupuesto

1 Sobre el presupuesto

En el presupuesto se han recogido todos los gastos asociados al desarrollo. En referencia al hardware, se incluye la compra de todos los componentes que han sido utilizados y en lo referente al software, se incluyen las licencias de los programas elegidos para el desarrollo del código la amortización del ordenador usado.

2 Materiales

Todos los precios que recoge el presente documento en relación con los componentes utilizados datan de julio de 2022.

Para obtener un presupuesto más exacto, se debe calcular la amortización de los equipos con un alto coste y que no son utilizados únicamente para este proyecto. En este caso, el único ítem que cumple con estas características es el Portátil MSI Modelo GV62 8RD. Para calcular la vida útil del ordenador se multiplica el número de horas diarias de uso por los 365 días del año y por la vida estimada en años del producto. En este caso:

Unidad	Descripción	Coste Equipo	Vida útil (h)	Cantidad (h)	Precio	Total
U	Portátil MSI Modelo GV62 8RD	1,200.00 €	15000	300	0.080 €	24.00 €

Este precio se incluye en la partida de materiales. Nótese que se han incluido entradas para las licencias del software de Arduino y de Code Composer Studio para ser exhaustivos, pero estos programas son gratuitos.

Materiales				
Uds	Nombre	Cantidad	Precio (€)/U	Total
U	Kit Robot	1	137.93 €	137.93 €
U	BOOSTXL-EDUMKII	1	28.52 €	28.52 €
U	MSP-EXP432P401R	1	24.14 €	24.14 €
U	Módulo HC-05	1	7.53 €	7.53 €
U	Módulo HC-06	1	4.20 €	4.20 €
U	Bolsa x10 Cables Hembra-Hembra Dupont	1	2.13 €	2.13 €
U	Licencia IDE Arduino	1	- €	- €
U	Licencia Code Composer Studio	1	- €	- €
U	Portátil MSI Modelo GV62 8RD	1	24.00 €	24.00 €
	Subtotal			**228.45 €**

El coste total de los materiales asciende a **228.45 €**

3 Mano de obra

Este apartado supone el gasto principal debido al gran porcentaje respecto al total del proyecto que supone el desarrollo del código y la prueba de este. El coste por hora se ha estimado en 15.00€ basándose en el sueldo medio de un recién graduado contratado como ingeniero.

Mano de obra				
Uds	**Nombre**	**Cantidad**	**Precio (€)/U**	**Total**
h	Planificación	30	15.00 €	450.00 €
h	Programación	155	15.00 €	2,325.00 €
h	Ensamblaje	5	15.00 €	75.00 €
h	Prueba	110	15.00 €	1,650.00 €
	Subtotal			**4,500.00 €**

El coste total de la mano de obra asciende a 4,500.00€

4 Coste del proyecto

En este apartado se exponen los costes del proyecto calculados en los apartados anteriores y el coste total.

Coste del proyecto	
Descripción	**Coste**
Materiales	228.45 €
Mano de obra	4,500.00 €
Subtotal	**4,728.45 €**
Beneficio Industrial (6%)	283.71 €
Gastos Adicionales (10%)	472.85 €
Total Antes de impuestos	**5,485.00 €**
IVA(21%)	1,151.85 €
Coste Totales:	**6,636.85 €**

La elaboración completa del proyecto con IVA incluido supone un gasto total de: **SEIS MIL SEICIENTOS TREINTA Y SEIS EUROS CON OCHENTA Y CINCO CÉNTIMOS (6,636.85€).**

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de Valéncia

Escuela Técnica Superior de Ingeniería del Diseño

Comunicación inalámbrica entre microcontroladores para controlar un robot de forma remota.

4. PLANOS

Autor: Montesinos Berenguer, Daniel

Tutor: Rodríguez Ballester, Francisco

Curso Académico: 2021/2022

Sobre los planos

Debido a que el objetivo de este proyecto ha sido realizar la comunicación de dos sistemas independientes disponibles en el mercado y no el diseño de estos ni de ningún componente no hay planos que incluir en este apartado del documento.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de Valéncia

Escuela Técnica Superior de Ingeniería del Diseño

Comunicación inalámbrica entre microcontroladores para controlar un robot de forma remota.

5. ANEXOS

Autor: Montesinos Berenguer, Daniel

Tutor: Rodríguez Ballester, Francisco

Curso Académico: 2021/2022

Índice Anexos

1. Códigos

1.1 Código mando

```
/* Standard includes */
#include <ti/devices/msp432p4xx/inc/msp.h>
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#include <ti/grlib/grlib.h>
#include "LcdDriver/Crystalfontz128x128_ST7735.h"
#include <stdint.h>
/*#include <>
#include "css/startup_msp432p401r_css.c"*/
/* MCU include */
#include <ti/drivers/gpio/GPIOMSP432.h>
#include <ti/devices/msp432p4xx/driverlib/uart.h>
//#include "main.h"

/*Inits*/
void initadc(void);
void initdysplay(void);
void initsystem(void);
void inituart(void);
void initpin(void);
/* display*/
void writedisplay(char word[20]);
void writenumber (uint16_t number);
void clearlcd(void);
/*Peripherals*/
void readjoystick( uint16_t v[2]);
void mapread(uint16_t v[2],uint_fast8_t k[2]);
uint8_t readbutton(uint_fast8_t Port, uint_fast16_t Pin);
void ledon(uint_fast8_t Port, uint_fast16_t Pin);
void ledoff(uint_fast8_t Port, uint_fast16_t Pin);
/**/
void menu1(void);
void menu2(void);
void menu3(void);
void readpos(void);
/*Send data*/
void senddata(uint_fast8_t data);
/*Move*/
void move(void);
/*Handlers*/
void PORT3_IRQHandler(void);
void PORT5_IRQHandler(void);
void SysTick_Handler(void);
/* */
void delay(const uint32_t microseconds); //microseconds
```

```
uint32_t numbertics (const uint32_t microseconds);
void setquotient(const uint32_t tics);
void setremainder( const uint32_t tics);
uint8_t getquotient(void);
uint32_t getremainder(void);
/* Graphic library context */
Graphics_Context g_sContext;

/*Defines*/
#define Port5   GPIO_PORT_P5
#define Pin1    GPIO_PIN1
#define Port3   GPIO_PORT_P3
#define Pin5    GPIO_PIN5
#define Port1   GPIO_PORT_P1
#define Pin0    GPIO_PIN0
#define COUNTS  15000000
/* ADC results buffer */
//static uint16_t resultsBuffer[2];
//uint_fast8_t mapresults[2];
uint8_t pos=1;
uint8_t y=10;
uint8_t menu=1;
uint16_t read1=1, read2=1;
volatile uint8_t aux=0;
uint32_t remainder;
uint8_t quotient;
uint8_t change=1;
char string[15];

int main(void)
 {
    WDT_A_holdTimer();              /* Stop watchdog timer */
    initsystem();
    initdysplay();
    initadc();
    inituart();
    initpin();
    Interrupt_enableMaster();

    ledoff(Port1,Pin0);
    read1=1;
    read2=1;

    while(1)
    {
        switch (menu)
        {
        case 1:
            delay(1000);
```

```
            menu1();
            break;
        case 2:
            delay(1000);
            menu2();
            break;
        case 3:
            delay(1000);
            menu3();
            break;
        }
    }
 }


void initsystem(void)
{
    /* Halting WDT and disabling master interrupts */
    MAP_WDT_A_holdTimer();
    MAP_Interrupt_disableMaster();

    /* Set the core voltage level to VCORE1 */
    MAP_PCM_setCoreVoltageLevel(PCM_VCORE1);

    /* Set 2 flash wait states for Flash bank 0 and 1*/
    MAP_FlashCtl_setWaitState(FLASH_BANK0, 2);
    MAP_FlashCtl_setWaitState(FLASH_BANK1, 2);

    /* Initializes Clock System */
    MAP_CS_setDCOCenteredFrequency(CS_DCO_FREQUENCY_48);
    MAP_CS_initClockSignal(CS_MCLK, CS_DCOCLK_SELECT,
CS_CLOCK_DIVIDER_1);
    MAP_CS_initClockSignal(CS_HSMCLK, CS_DCOCLK_SELECT,
CS_CLOCK_DIVIDER_1);
    MAP_CS_initClockSignal(CS_SMCLK, CS_DCOCLK_SELECT,
CS_CLOCK_DIVIDER_1);
    MAP_CS_initClockSignal(CS_ACLK, CS_REFOCLK_SELECT,
CS_CLOCK_DIVIDER_1);



}

void initpin(void)
{
    GPIO_setAsOutputPin(Port1, Pin0); // Pin1 output
    GPIO_setAsInputPinWithPullUpResistor(Port3, Pin5); //Button
1 --> input || 1 Not pressed --> 0 presses
```

```
    GPIO_setAsInputPinWithPullUpResistor(Port5, Pin1); //Button
2 --> input || 1 Not pressed --> 0 pressed

    GPIO_enableInterrupt(Port3, Pin5); //Init Interrupt
    GPIO_enableInterrupt(Port5, Pin1); //Init Interrupt
    GPIO_clearInterruptFlag(Port3, Pin5); //Clear flag
    GPIO_clearInterruptFlag(Port5, Pin1); // Clear flag
    GPIO_interruptEdgeSelect(Port3, Pin5,
GPIO_LOW_TO_HIGH_TRANSITION); // 0 --> 1
    GPIO_interruptEdgeSelect(Port5, Pin1,
GPIO_LOW_TO_HIGH_TRANSITION); // 0 --> 1

    GPIO_registerInterrupt(Port3,PORT3_IRQHandler);
    GPIO_registerInterrupt(Port5,PORT5_IRQHandler);

    /*Init SysTick*/
    SysTick_enableModule();

    SysTick_enableInterrupt();

    SysTick_registerInterrupt(SysTick_Handler);

}


void initdysplay(void)
{
    /* Initializes display */
    Crystalfontz128x128_Init();

    /* Set default screen orientation */
    Crystalfontz128x128_SetOrientation(LCD_ORIENTATION_UP);

    /* Initializes graphics context */
    Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128,
&g_sCrystalfontz128x128_funcs);
    Graphics_setForegroundColor(&g_sContext,
GRAPHICS_COLOR_RED);
    Graphics_setBackgroundColor(&g_sContext,
GRAPHICS_COLOR_WHITE);
    GrContextFontSet(&g_sContext, &g_sFontFixed6x8);
    Graphics_clearDisplay(&g_sContext);
}

void initadc(void)
{

    /* Configures Pin 6.0 and 4.4 as ADC input */
```

```
    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6,
GPIO_PIN0, GPIO_TERTIARY_MODULE_FUNCTION);
    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P4,
GPIO_PIN4, GPIO_TERTIARY_MODULE_FUNCTION);

    /* Initializing ADC (ADCOSC/64/8) */
    MAP_ADC14_enableModule();
    MAP_ADC14_initModule(ADC_CLOCKSOURCE_ADCOSC,
ADC_PREDIVIDER_1, ADC_DIVIDER_1, 0);

    /* Configuring ADC Memory (ADC_MEM0 - ADC_MEM1 (A15, A9)
with repeat)
     * with internal 2.5v reference */
    MAP_ADC14_configureMultiSequenceMode(ADC_MEM0, ADC_MEM1,
true);
    MAP_ADC14_configureConversionMemory(ADC_MEM0,

ADC_VREFPOS_AVCC_VREFNEG_VSS,
                                        ADC_INPUT_A15,
ADC_NONDIFFERENTIAL_INPUTS);

    MAP_ADC14_configureConversionMemory(ADC_MEM1,

ADC_VREFPOS_AVCC_VREFNEG_VSS,
                                        ADC_INPUT_A9,
ADC_NONDIFFERENTIAL_INPUTS);

    /* Enabling the interrupt when a conversion on channel 1
(end of sequence)
     *  is complete and enabling conversions */
    //MAP_ADC14_enableInterrupt(ADC_INT1);

    /* Enabling Interrupts */
    //MAP_Interrupt_enableInterrupt(INT_ADC14);
    //MAP_Interrupt_enableMaster();

    /* Setting up the sample timer to automatically step through
the sequence
     * convert.
     */
    MAP_ADC14_enableSampleTimer(ADC_AUTOMATIC_ITERATION);

    /* Triggering the start of the sample */
    MAP_ADC14_enableConversion();
    MAP_ADC14_toggleConversionTrigger();


}
```

```
void inituart(void)
{

    GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P3,
GPIO_PIN2, GPIO_PRIMARY_MODULE_FUNCTION);
    GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P3,
GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);

    const eUSCI_UART_ConfigV1 uartConfig =
    {
     EUSCI_A_UART_CLOCKSOURCE_SMCLK, // SMCLK Clock Source
     312, // BRDIV = 78
     8, // UCxBRF = 2
     0, // UCxBRS = 0
     EUSCI_A_UART_NO_PARITY, // No Parity
     EUSCI_A_UART_LSB_FIRST, // LSB First
     EUSCI_A_UART_ONE_STOP_BIT, // One stop bit
     EUSCI_A_UART_MODE, // UART mode
     EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION, //
Oversampling
     EUSCI_A_UART_8_BIT_LEN // DataLegth
    };

    /* Configuring UART Module */
    MAP_UART_initModule(EUSCI_A2_BASE, &uartConfig);
    /* Enable UART module */
    MAP_UART_enableModule(EUSCI_A2_BASE);

}

void writedisplay(char word[20])
{
    //char string[15];
    sprintf(string, "%s", word);
    Graphics_drawStringCentered(&g_sContext,
                                (int8_t *)string,
                                AUTO_STRING_LENGTH,
                                64,
                                y,
                                OPAQUE_TEXT);
    y=y+10;
}

void writenumber(uint16_t number )
{
    // char string[15];
    sprintf(string, "%5d", number);
    Graphics_drawStringCentered(&g_sContext,
                                (int8_t *)string,
```

```
                                    AUTO_STRING_LENGTH ,
                                    64,
                                    y,
                                    OPAQUE_TEXT);
    y=y+10;
}

void readjoystick(uint16_t resultsBuffer[2])
{
    resultsBuffer[0] = ADC14_getResult(ADC_MEM0); //Pos X
    resultsBuffer[1] = ADC14_getResult(ADC_MEM1); //Pos Y
    //printf("%d \n", resultsBuffer[0]);
    //printf("%d \n", resultsBuffer[1]);
        //printf("%d \n",k[1]);

}


void clearlcd(void)
{
    y=0;
    uint16_t k=0;
    for ( k=0;k<=13;k++)
    {writedisplay("                  ");}//18
    y=10;
}

uint8_t readbutton(uint_fast8_t Port, uint_fast16_t Pin)
{
    return (GPIO_getInputPinValue(Port, Pin));
}

void ledon(uint_fast8_t Port, uint_fast16_t Pin)
{
    GPIO_setOutputHighOnPin(Port, Pin);
}

void ledoff(uint_fast8_t Port, uint_fast16_t Pin)
{
    GPIO_setOutputLowOnPin(Port, Pin);
}

void menu1(void)
{
    y=60;

    writedisplay("Press button A");
    change = 1;
    while(read1 == 1  )
```

```
    {
        ledon(Port1,Pin0);
    }

    clearlcd();

    menu=2;
    read1=1;
}

void menu2(void)
{
    ledoff(Port1,Pin0);
    while(read1 == 1 && read2==1)
    {
        senddata(255);
        senddata(255);
        y=50;

        switch (pos){
        case 1:
            writedisplay("->A.Free");
            writedisplay("B.Without shock ");
            writedisplay("C.Approximation ");
            break;
        case 2:
            writedisplay("A.Free ");
            writedisplay("->B.Without shock ");
            writedisplay("C.Approximation  ");
            break;
        case 3:
            writedisplay("A.Free  ");
            writedisplay("B.Without shock  ");
            writedisplay("->C.Approximation ");
            break;
        }

        readpos();
        delay(150);
        change = 1;
    }
    clearlcd();
    if(read1==0)
    {menu=3;read1=1;}
    else
    {menu=1;read2=1;}
}
```

```
void menu3(void)
{
    while (read2==1)
    {
        change = 1;
        y=80;
        ledon(Port1,Pin0);
        switch (pos)
        {
        case 1:
            delay(2000000);
            senddata(1);
            writedisplay("Free mode ");
            move();
            break;
        case 2:
            delay(2000000);
            senddata(2);
            writedisplay("Without shock ");
            move();
            break;
        case 3:
            delay(2000000);
            senddata(3);
            writedisplay("Approximation");
            move();
            break;
        }
        senddata(255);
        senddata(255);
    }

    menu=2;
    read2=1;

    clearlcd();
}

void  readpos(void)
{
    uint16_t value[2];
    readjoystick(value);

    if(value[1]>10000)
    {pos--;}
    if(value[1]<6000)
    {pos++;}
    if(pos<1)
    {pos=1;}
```

```
    if(pos>3)
    {pos=3;}

}


void senddata(uint_fast8_t data)
{
    UART_transmitData(EUSCI_A2_BASE,data);
}

void mapread(uint16_t v[2],uint_fast8_t k[2])
{

    k[0]=(v[0]*255)/16400;
    k[1]=(v[1]*255)/16400;

}

void move(void)
{
    static uint16_t results[2];
    uint_fast8_t k1;
    uint_fast8_t k2;
    uint_fast8_t mapresults[2];
    while(read2==1)
    {
        delay(100);
        readjoystick(results);
        mapread(results,mapresults);
        k1=mapresults[0];
        k2=mapresults[1];
        senddata(k1);
        delay(100);
        senddata(k2);
    }

}

void PORT5_IRQHandler(void) //Button A
{
    GPIO_disableInterrupt(Port5,Pin1);
    GPIO_clearInterruptFlag(Port5,Pin1);
    if(change == 1)
    {
        read1 = 0;
        change = 0;
    }
    GPIO_enableInterrupt(Port5,Pin1);
```

```
}

void PORT3_IRQHandler(void) //Button B
{
    GPIO_disableInterrupt(Port3,Pin5);
    GPIO_clearInterruptFlag(Port3,Pin5);
    if(change==1)
    {
        read2 = 0;
        change = 0;
    }
    GPIO_enableInterrupt(Port3,Pin5);
}

void SysTick_Handler(void)
{
    SysTick_disableInterrupt();
    //GPIO_toggleOutputOnPin(Port1,Pin0);
    //printf("%d \n",SysTick_getPeriod());
    aux ++;
    SysTick_enableInterrupt();
}

void delay(const uint32_t microseconds)
{
    // Time in microseconds
    //setquotient(tics);
    //setremainder(tics);
    // GPIO_disableInterrupt(Port3,Pin5);
    //GPIO_disableInterrupt(Port5,Pin1);
    setquotient(numbertics(microseconds));
    setremainder(numbertics(microseconds));
    if(numbertics(microseconds)>COUNTS)
    {SysTick_setPeriod (COUNTS);}
    else
    {SysTick_setPeriod (numbertics(microseconds));}

    while(aux!=getquotient()+1)
    {
       // ledoff(Port1,Pin0);
        if(aux==getquotient())
        {
            SysTick_setPeriod (getremainder());
        }
        if(read1 == 0 || read2 == 0)
        {
            aux=getquotient()+1;
        }
    }
```

```
    //GPIO_enableInterrupt(Port5,Pin1);
    //GPIO_enableInterrupt(Port3,Pin5);
    aux = 0;
    /* read1=1;
    read2=1;*/
}

uint32_t numbertics(const uint32_t microseconds)
{
    //  printf("%d\n ",microseconds*48);
    return (microseconds*48);

}

void setquotient(const uint32_t tics)
{
    quotient = tics/COUNTS;
}
void setremainder( const uint32_t tics)
{
    remainder = tics%COUNTS;
}


uint8_t getquotient(void)
{
    return quotient;
}

uint32_t getremainder(void)
{
    return remainder;
}
```

1.2 Código Robot

```
#define US 2
//Right wheel
#define Right_dir 12
#define Right_pwm 11
//Left wheel
#define Left_dir 8
#define Left_pwm 3

float DISTANCE = 15.0;
uint8_t direct;
bool stop=false;
uint8_t sp, sp2, mode; //speed
float distance;
byte b[8] = {126, 126, 0, 0, 0, 0, 0, 0};

typedef struct
{
  uint8_t dir_pin;
  uint8_t pwm_pin;
  uint8_t forward_level;

} Wheel;

Wheel Left_Wheel = {12, 11,LOW};
Wheel Right_Wheel = {8, 3, HIGH};

void initwheel(const Wheel side)
{
  pinMode(side.dir_pin, OUTPUT);
  digitalWrite(side.pwm_pin, LOW);
}


float measuredistance(void)
{
  pinMode(US, OUTPUT);
  digitalWrite(US, LOW);
  delayMicroseconds(2);
  digitalWrite(US, HIGH);
  delayMicroseconds(5);
  digitalWrite(US, LOW);
  pinMode(US, INPUT);
  unsigned long tm = pulseIn(US, HIGH);
  float distance = tm / 58.0;
  delayMicroseconds(50);
  //Serial.println(distance);
  return distance;
}

void forward(int pwm, const Wheel side)
{
  if (pwm < 0)
```

```
  {
    digitalWrite(side.dir_pin, !side.forward_level);
    pwm = -pwm;
  }
  else
  {
    digitalWrite(side.dir_pin, side.forward_level);
  }
  analogWrite(side.pwm_pin, pwm > 255 ? 255 : pwm);
}

void backward(int pwm, const Wheel side)
{
  forward(-pwm, side);
}

void readSerial(byte b[8])
{
  if (Serial.available() > 0)
  {
    Serial.readBytes(b, 8);
  }
}

void modeWS(int y, int x)
{
  //Serial.println("ModeWS");
  byte b[8] = {126, 126, 0, 0, 0, 0, 0, 0};
  float distance = DISTANCE + 1 ;
  //if((ref[0]>100 && ref[0]<155) && (ref[1]<155 && ref
[1]>100))//quieto

  while ((y != 255 || x != 255 ) )
  {
    readSerial(b);
    y = b[0];
    x = b[1];

   // Serial.println(y);
   // Serial.println(x);
    if ((y > 100 && y < 155) && (x < 155 && x > 100)) //quieto
      {
    //    Serial.println("Quieto");
        forward(0,Right_Wheel);
          forward(0,Left_Wheel);
      }
    if (direct == 1)
    {
          if (y > 155 && (x < 155 && x > 100)) //direct
          {
           // Serial.println("direct");
             sp=map(y,155,255,0,155);
              forward(sp,Right_Wheel);
             //sp2=map(y,155,255,0,255);
              forward(sp,Left_Wheel);
            // Serial.println(sp);
```

```
108.          }
109.
110.          if (x > 155 && (y > 100 && y < 155)) //DERECHA // Asi no
    va 100<ref[0]<155
111.          {
112.            //Serial.println("Derecha");
113.            sp=map(x,155,255,0,155);
114.              forward(sp,Right_Wheel);
115.              forward(0,Left_Wheel);
116.            // Serial.println(sp);
117.
118.          }
119.
120.          if (x < 100 && (y > 100 && y < 155)) //IZQUIERDA
121.          {
122.          // Serial.println("Izquierda");
123.              sp=map(x,0,100,155,0);
124.              forward(0,Right_Wheel);
125.              forward(sp,Left_Wheel);
126.            //Serial.println(sp);
127.          }
128.
129.          if (y > 155 && x > 155) //cuadrante 1
130.          {
131.            //Serial.println("Cuadrante 1");
132.              sp=map(y,155,255,0,155);
133.              forward(sp,Right_Wheel);
134.              sp2=map(x,0,100,155,0);
135.              forward(sp,Left_Wheel);
136.            //Serial.println(sp);
137.          }
138.
139.          if (y > 155 && x < 100) //cuadrante 2
140.          {
141.            //Serial.println("Cuadrante 2");
142.            sp=map(y,155,255,0,155);
143.              forward(sp,Left_Wheel);
144.              sp2=map(x,155,255,0,155);
145.              forward(sp2,Right_Wheel);
146.            //Serial.println(sp);
147.          }
148.        }
149.        if (y < 100 && (x < 155 && x > 100))
150.        {
151.          //Serial.println("Atrás");
152.          sp=map(y,0,100,155,0);
153.            backward(sp,Right_Wheel);
154.            //sp2=map(x,0,100,255,0);
155.            backward(sp,Left_Wheel);
156.          //Serial.println(sp);
157.        }
158.
159.        if (y < 100 && x < 100)
160.        {
161.          //Serial.println("Cuadrante 3");
162.           sp=map(y,0,100,155,0);
```

```
            backward(sp,Right_Wheel);
            sp2=map(x,0,100,155,0);
            backward(sp2,Left_Wheel);
          //Serial.println(sp);

        }

        if (y < 100 && x > 155)
        {
          //Serial.println("Cuadrante 4");
          sp=map(y,0,100,155,0);
            backward(sp,Right_Wheel);
            sp2=map(x,155,255,0,155);
            backward(sp2,Left_Wheel);
            //Serial.println(sp);
        }

        distance = measuredistance();
        if (distance < DISTANCE )
        {
          direct = 0;
          if (stop == false)
          {
              forward(0,Right_Wheel);
              forward(0,Left_Wheel);
              stop = true;
          }
        }
        else
        {
          if (distance > DISTANCE)
          {
              direct = 1;
             stop=false;
          }
        }
        readSerial(b);
        y = b[0];
        x = b[1];
      }
      delay(250);
    }

    void modeAprox(int y, int x)
    {
      //Serial.println("ModeAprox");
      float dist = DISTANCE;
      float cont = 0.0;
      byte b[8] = {126, 126, 0, 0, 0, 0, 0, 0};
      float distance = DISTANCE + 1 ;
      //if((ref[0]>100 && ref[0]<155) && (ref[1]<155 && ref
[1]>100))//quieto
      while ((y != 255 || x != 255 ) )
      {
        readSerial(b);
        y = b[0] ;
```

```
218.         x = b[1] ;
219.
220.         if ((y > 100 && y < 155) && (x < 155 && x > 100)) //quieto
221.           {
222.            // Serial.println("Quieto");
223.             forward(0,Right_Wheel);
224.               forward(0,Left_Wheel);
225.           }
226.         if (direct == 1)
227.         {
228.
229.
230.           if (y > 155 && (x < 155 && x > 100)) //direct
231.           {
232.            // Serial.println("direct");
233.             sp=map(y,155,255,0,255);
234.               forward(y,Right_Wheel);
235.              //sp2=map(x,155,255,0,255);
236.               forward(x,Left_Wheel);
237.             // Serial.println(sp);
238.
239.           }
240.
241.           if (x > 155 && (y > 100 && y < 155)) //DERECHA // Asi no
    va 100<ref[0]<155
242.           {
243.          //   Serial.println("Derecha");
244.             sp=map(x,155,255,0,155);
245.               forward(sp,Right_Wheel);
246.               forward(0,Left_Wheel);
247.             // Serial.println(sp);
248.
249.           }
250.
251.           if (x < 100 && (y > 100 && y < 155)) //IZQUIERDA
252.           {
253.            // Serial.println("Izquierda");
254.               sp=map(x,0,100,155,0);
255.               forward(0,Right_Wheel);
256.               forward(sp,Left_Wheel);
257.             //Serial.println(sp);
258.           }
259.
260.           if (y > 155 && x > 155) //cuadrante 1
261.           {
262.               sp=map(y,155,255,0,155);
263.               forward(sp,Right_Wheel);
264.               sp2=map(x,0,100,155,0);
265.               forward(sp2,Left_Wheel);
266.           }
267.
268.           if (y > 155 && x < 100) //cuadrante 2
269.           {
270.            // Serial.println("Cuadrante 2");
271.             sp=map(y,155,255,0,155);
272.               forward(sp,Left_Wheel);
```

```
            sp2=map(x,155,255,0,155);
            forward(sp2,Right_Wheel);

          //Serial.println(sp);

        }
      }
      if (y < 100 && (x < 155 && x > 100))
      {
       // Serial.println("Atrás");
       sp=map(y,0,100,155,0);
          backward(sp,Right_Wheel);
          //sp2=map(x,0,100,255,0);
          backward(sp,Left_Wheel);
        //Serial.println(sp);
      }

      if (y < 100 && x < 100)
      {
      //  Serial.println("Cuadrante 3");
         sp=map(y,0,100,155,0);
          backward(sp,Right_Wheel);
          sp2=map(x,0,100,155,0);
          backward(sp2,Left_Wheel);
        //Serial.println(sp);

      }

      if (y < 100 && x > 155)
      {
      // Serial.println("Cuadrante 4");
        sp=map(y,0,100,155,0);
          backward(sp,Right_Wheel);
          sp2=map(x,155,255,0,155);
          backward(sp2,Left_Wheel);
          //Serial.println(sp);
      }

      distance = measuredistance();

      if (distance < dist )
      {
        forward(0,Right_Wheel);
            forward(0,Left_Wheel);
        delay(300);
        dist=dist-3;
      }
       if (distance < 2.99)
      {
        direct = 0;
         if (stop == false)
        {
            forward(0,Right_Wheel);
            forward(0,Left_Wheel);
            stop = true;
        }
```

```
        }
        else if (distance > 3.00)
        {
            direct = 1;
            stop = false;
        }
        readSerial(b);
        y = b[0];
        x = b[1];
      }
      delay(250);
    }


    void modefree(int y, int x)
    {
      //Serial.println("Free");
      byte b[8] = {126, 126, 0, 0, 0, 0, 0, 0};

      while ((y != 255) || (x != 255))
      {
        readSerial(b);
        y = b[0] ;
        x = b[1] ;

        if ((y > 100 && y < 155) && (x < 155 && x > 100))//quieto
          {
           // Serial.println("Quieto");
            forward(0,Right_Wheel);
              forward(0,Left_Wheel);
          }


          if (y > 155 && (x < 155 && x > 100)) //direct
          {
           // Serial.println("direct");
            sp=map(y,155,255,0,255);
              forward(155,Right_Wheel);
             //sp2=map(x,155,255,0,255);
              forward(155,Left_Wheel);
            // Serial.println(sp);

          }

          if (x > 155 && (y > 100 && y < 155)) //DERECHA //
          {
         //   Serial.println("Derecha");
            sp=map(x,155,255,0,155);
              forward(sp,Right_Wheel);
              forward(0,Left_Wheel);
            // Serial.println(sp);

          }

          if (x < 100 && (y > 100 && y < 155)) //IZQUIERDA
          {
```

```
385.            // Serial.println("Izquierda");
386.             sp=map(y,155,255,0,155);
387.               forward(sp,Right_Wheel);
388.               sp2=map(x,0,100,155,0);
389.               forward(sp2,Left_Wheel);
390.             //Serial.println(sp);
391.           }
392.
393.           if (y > 155 && x > 155) //cuadrante 1
394.           {
395.            // Serial.println("Cuadrante 1");
396.              sp=map(y,155,255,0,155);
397.               forward(sp,Left_Wheel);
398.               sp2=map(x,155,255,0,155);
399.               forward(sp2,Right_Wheel);
400.             //Serial.println(sp);
401.           }
402.
403.           if (y > 155 && x < 100) //cuadrante 2
404.           {
405.            // Serial.println("Cuadrante 2");
406.                sp=map(y,155,255,0,155);
407.               forward(sp,Left_Wheel);
408.               sp2=map(x,155,255,0,155);
409.               forward(sp2,Right_Wheel);
410.
411.             //Serial.println(sp);
412.
413.           }
414.
415.         if (y < 100 && (x < 155 && x > 100))
416.         {
417.          // Serial.println("Atrás");
418.          sp=map(y,0,100,155,0);
419.             backward(sp,Right_Wheel);
420.             //sp2=map(x,0,100,255,0);
421.             backward(sp,Left_Wheel);
422.           //Serial.println(sp);
423.         }
424.
425.         if (y < 100 && x < 100)
426.         {
427.         //  Serial.println("Cuadrante 3");
428.            sp=map(y,0,100,155,0);
429.             backward(sp,Right_Wheel);
430.             sp2=map(x,0,100,155,0);
431.             backward(sp2,Left_Wheel);
432.           //Serial.println(sp);
433.
434.         }
435.
436.         if (y < 100 && x > 155)
437.         {
438.         //  Serial.println("Cuadrante 4");
439.           sp=map(y,0,100,155,0);
440.             backward(sp,Right_Wheel);
```

```
            sp2=map(x,155,255,0,155);
            backward(sp2,Left_Wheel);
            //Serial.println(sp);
        }
        readSerial(b);
        y = b[0] ;
        x = b[1] ;
      }
       delay(500);
    }


    void setup() {
      Serial.begin(9600);
      initwheel(Left_Wheel);
      initwheel(Right_Wheel);
      forward(0, Right_Wheel);
      forward(0, Left_Wheel);
      pinMode(7,OUTPUT);
    }

    void loop() {
      mode=0;
      readSerial(b);
      mode = b[0];
      switch (mode)
      {
        case 1:
          readSerial(b);
          modefree(b[0], b[1]);
          mode=0;
          forward(0,Right_Wheel);
          forward(0,Left_Wheel);
          break;
        case 2:
          readSerial(b);
          modeWS(b[0], b[1]);
          mode=0;
          forward(0,Right_Wheel);
          forward(0,Left_Wheel);
          break;
        case 3:
          readSerial(b);
          modeAprox(b[0], b[1]);
          mode=0;
          forward(0,Right_Wheel);
          forward(0,Left_Wheel);
          break;
        default:
          mode=0;
          break;
      }
    }
```