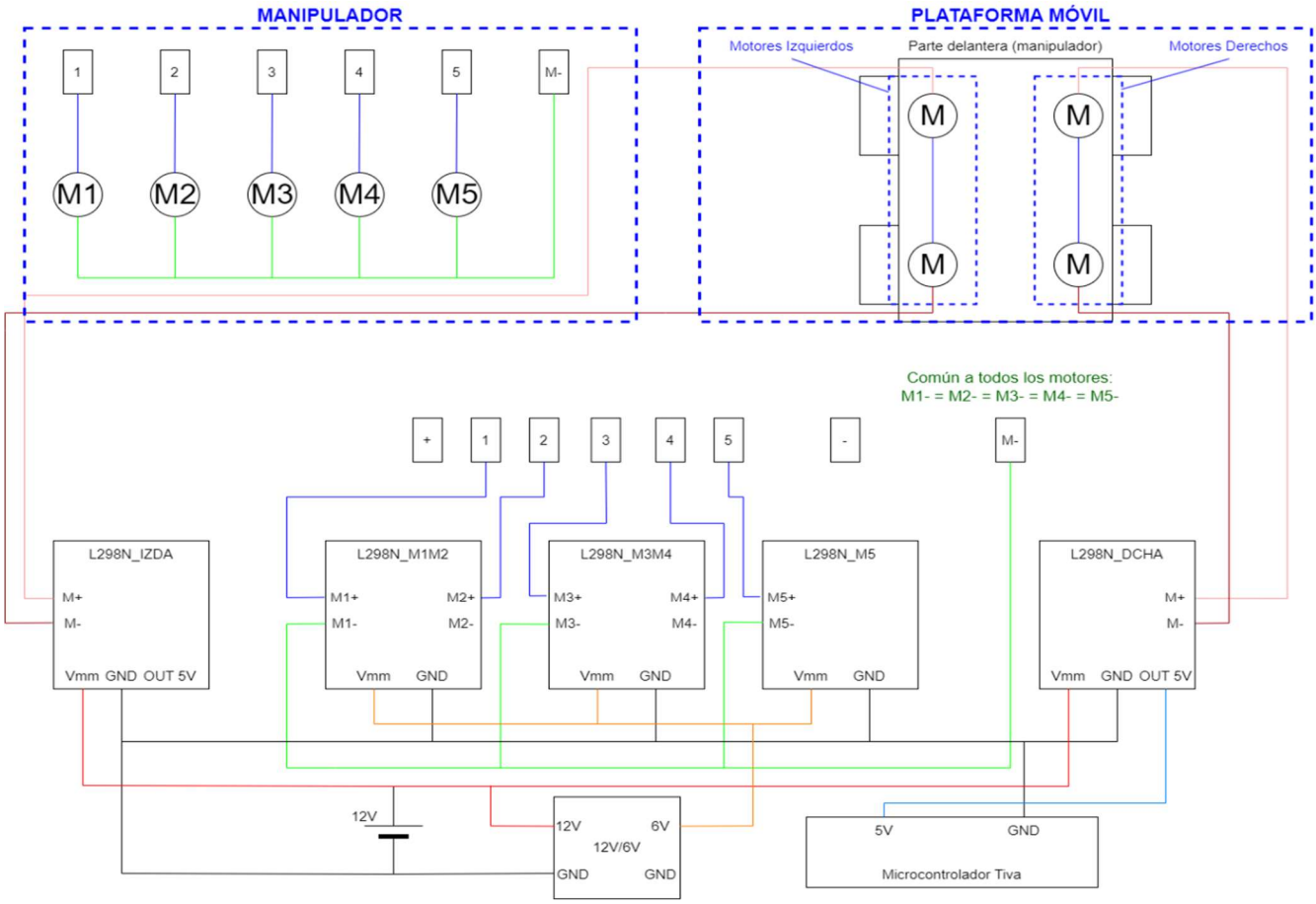


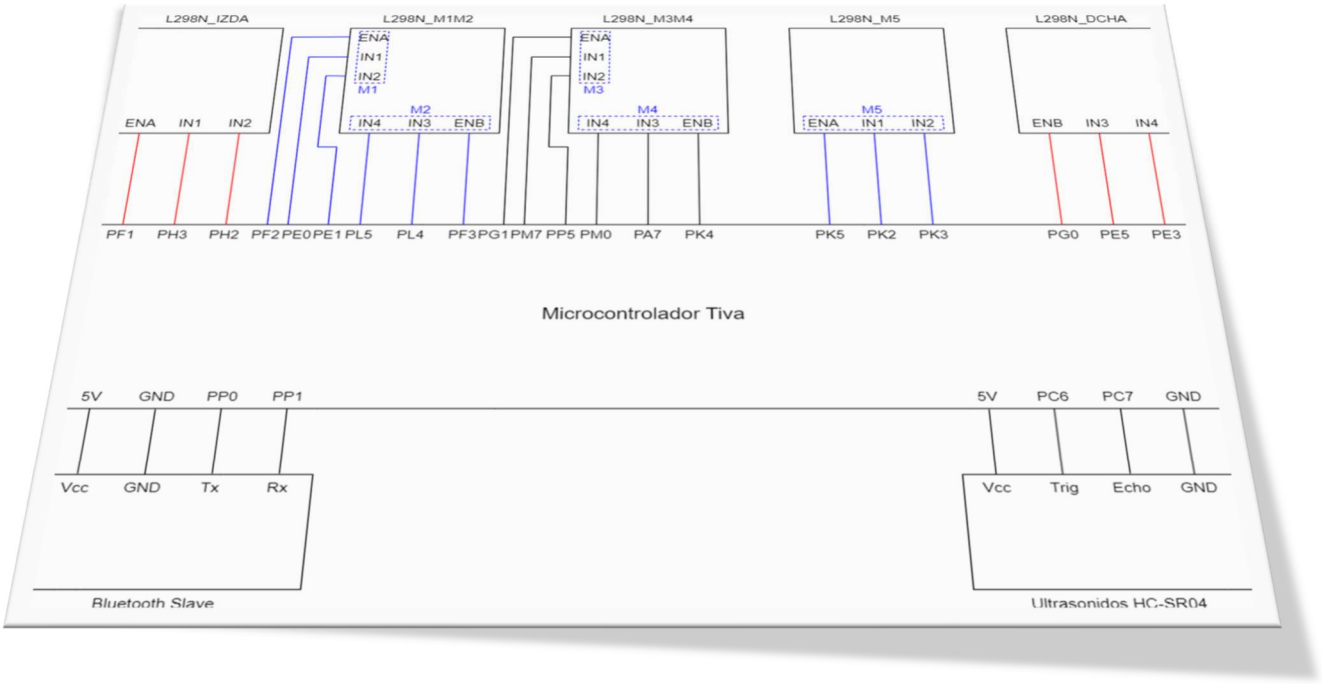
ROBOT RECOGEDOR: TRABAJO DE SEPA (MICROCONTROLADORES)

4º GIERM

Práctico robot de limpieza para el hogar que permite recoger objetos y enseres sin esfuerzo desde el sofá, utilizando un mando Bluetooth con pantalla para controlarlo de manera sencilla y conveniente.

José Antonio Heredia Muñoz
Arturo Renato Úbeda Quilón

ÍNDICE DE CONTENIDOS



1. MOTIVACIÓN DEL PROYECTO

El proyecto que proponemos es un robot teleoperable mediante Bluetooth, cuya finalidad es permitir a su usuario recoger los distintos enseres de su domicilio desde la comodidad de su sofá, sin necesidad de hacerlo por él mismo, operando el robot mediante un mando Bluetooth con una pantalla.

Nuestra motivación es ayudar a personas mayores a recoger su casa sin tener que realizar movimientos complejos como agacharse, desplazar muebles, o llegar a zonas del hogar de difícil acceso, todo ello sin ayuda.

2. ALCANCE DEL PROYECTO

En este proyecto, presentamos un robot prototipo que consta de un robot manipulador montado sobre una plataforma móvil, con el objetivo de ilustrar nuestra motivación. Aunque el prototipo actual tiene ciertas limitaciones: este robot puede levantar objetos de hasta 100g, como tapones de botella, paquetes de pañuelos o bolígrafos.

Para lograr una mayor cobertura de nuestra motivación, consideramos que sería beneficioso aumentar la capacidad del manipulador, permitiéndole desplazar muebles dentro del hogar. También hubiéramos deseado diseñar un robot manipulador personalizado con un mayor número de grados de libertad, lo que permitiría acceder a zonas de la casa que son más difíciles de alcanzar.

Otra ampliación interesante pasaría por un control cinemático y dinámico para permitir movimientos automáticos del brazo, así como definir trayectorias para su elemento terminal. Esto haría que la recogida de objetos fuera más rápida, precisa y cómoda para el usuario.

Lograr ese grado de automatización requeriría:

- Disponer de un **encoder en cada rueda y en cada articulación** del brazo, para determinar la posición del robot.
- **Sensores** de posición de **mayor precisión** (LiDAR, sensores infrarrojos de precisión, cámaras, etc), que además de determinar con mayor precisión la presencia de obstáculos, **nos podrían permitir incluso** un mapeado del hogar y un posicionamiento del robot simultáneos en dicho mapa (métodos de **SLAM**), de manera que **el robot operaría** de forma **totalmente autónoma**.
- **Baterías de mayor capacidad** para permitir una mayor autonomía de la operación.
- **Cámaras** para permitir una teleoperación desde cualquier otra parte de la casa, e incluso **módulos wifi,** para permitir dicha **teleoperación por internet**.

3. JUSTIFICACIÓN DEL USO DEL MICROCONTROLADOR DE LA ASIGNATURA EN EL PROYECTO

Dado que necesitamos implementar algoritmos de control para los distintos sensores y actuadores del robot, es necesaria la presencia de un microcontrolador. Por otro lado, también necesitamos construir un mando que realice la lectura de distintos controles, así como el manejo de una pantalla. Por tanto, nuestro proyecto realmente necesita usar dos microcontroladores.

Nos decantamos por el uso del TIVA TM4C1294 de Texas Instruments dado que cuenta con un gran número de pines digitales, así como 7 PWM's accesibles y configurables para el control de los motores del robot móvil y del manipulador.

También haremos uso de dos UART's de cada microcontrolador, una para monitorización del robot/mando vía puerto serie (PuTTY), y otra para la comunicación por Bluetooth (el mando será el master, y el robot el slave).

El microcontrolador elegido cuenta también con ADC's de 16 bits, que usaremos para la lectura de los joystick del mando. Por otro lado, el manejo de la pantalla permitirá el diseño de una HMI para mostrar ayudas para el control del robot.

4. DESARROLLO HARDWARE DEL PROYECTO

4.1. FUNCIONAMIENTO DEL MANIPULADOR

4.1.1. FUNCIONAMIENTO COMERCIAL DEL MANIPULADOR. INVERSIÓN DE GIRO MEDIANTE CONMUTADORES FÍSICOS

El robot manipulador empleado en el trabajo consta de 5 articulaciones, movidas por 5 motores de corriente continua de idénticas características. El control se realiza articulación a articulación mediante un mando a distancia, que internamente implementa unos conmutadores físicos.

A continuación, mostraremos un esquema simplificado del control que realiza el mando sobre uno de los motores:

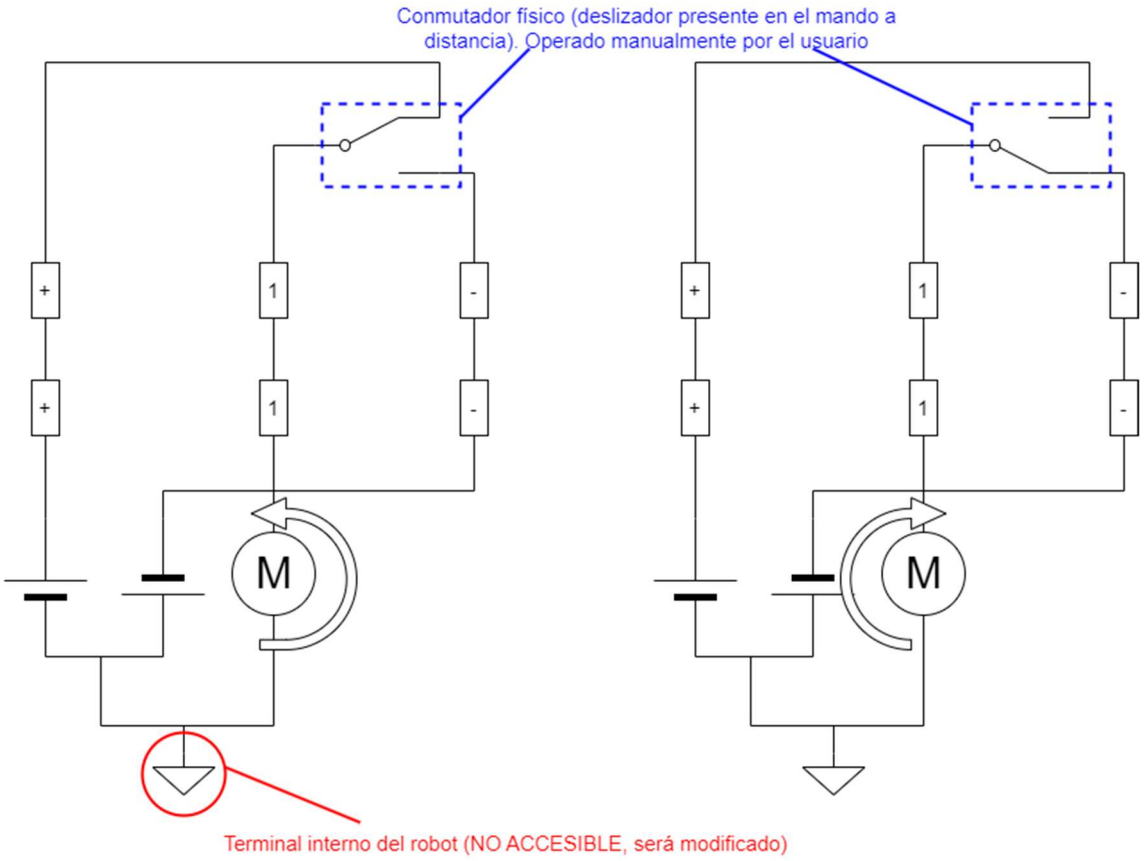


Figura 1. Esquema simplificado de la conexión entre el mando y uno de los motores del manipulador

Como podemos comprobar, la función del mando a distancia es permitir el paso de corriente. Según esté accionado el conmutador, conectará el terminal "1" con el "+", dando lugar a un giro antihorario del motor, o con el terminal "-", dando lugar a un giro horario.

4.2.2. ADAPTACIÓN DEL ESQUEMA ELÉCTRICO DEL MANIPULADOR

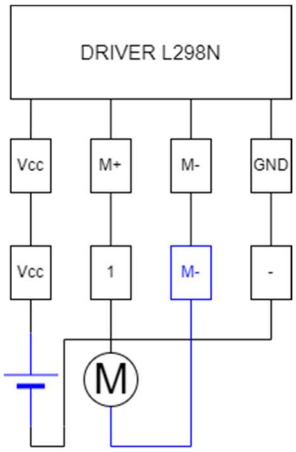


Figura 2. Modificaciones propuestas (azul) en la conexión entre el robot y su mando

La fuente de tensión del manipulador tiene una disposición incompatible para el uso del driver L298N (el anterior esquema trabaja sólo con 3 pines, conectando el terminal 1 del brazo con el + o el -, dado que el otro terminal del motor está conectado a una GND intermedia).

Se propone realizar la siguiente adaptación del anterior esquema eléctrico, proporcionando así los 4 pines que necesita el driver (M+, M-, Vcc, GND), excluyendo las señales digitales para el micro, que veremos más adelante.

Eliminaríamos la fuente doble de terminal intermedio, y haríamos accesible el otro terminal del motor, para hacer un control de velocidad y dirección de giro.

Para regular la velocidad y el sentido de giro de los motores usaremos un controlador de motores L298N.

Cada L298N puede manejar hasta dos motores. La figura 3 muestra el pinout y especificaciones de uso [1]. En [1], podemos encontrar una completa descripción del dispositivo.

Para adaptar la conexión entre el manipulador y su mando y conseguir la conexión propuesta en la página anterior para cada uno de sus motores, es necesario abrir la tapa del motor, y acceder a la PCB que se encarga de distribuir la potencia. Esta PCB tiene cinco tomas/puertos:

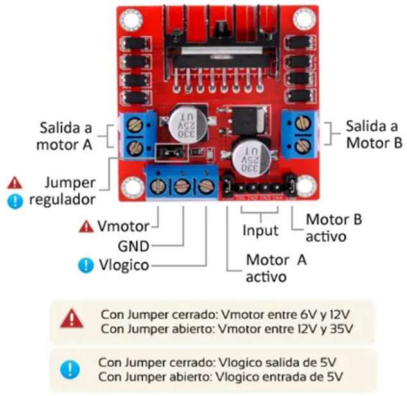


Figura 3. Esquema de uso del L298N. Fuente: [1]

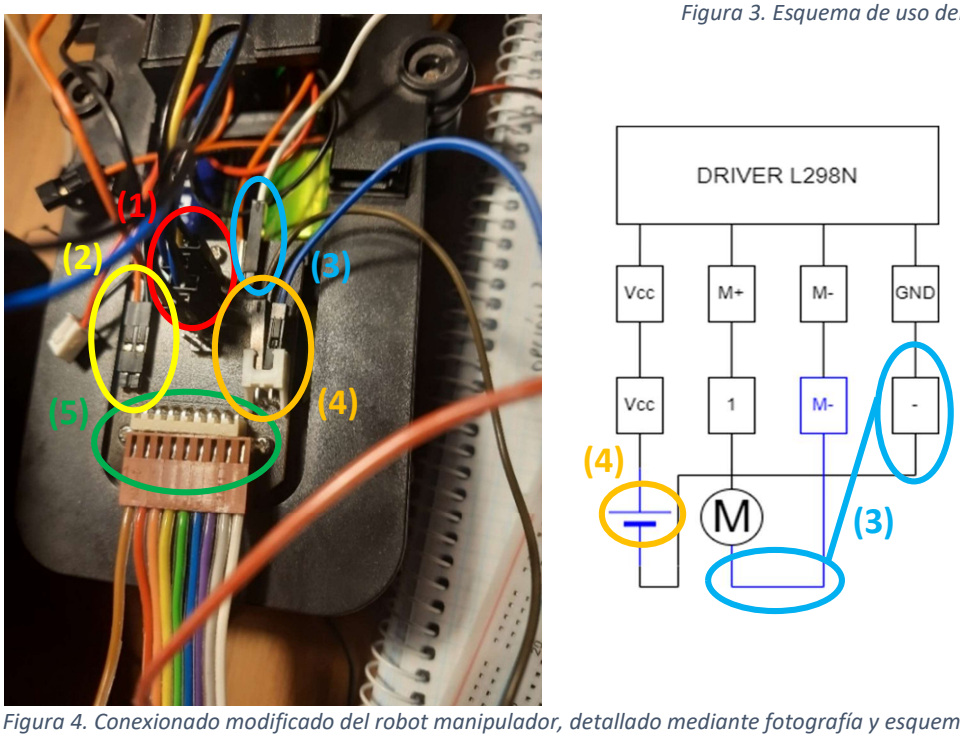


Figura 4. Conexionado modificado del robot manipulador, detallado mediante fotografía y esquemático

1. Peine de conexiones para los motores M1 al M4 (para los cables + y – que salen del motor físico) y una luz LED (que no hemos usado).
2. Conexión del motor M5 (que se hace conecta aparte, fuera del peine).
3. Conexión “GND”, ese terminal intermedio de la fuente doble, que a su vez es el terminal M- del motor, al que nos interesa acceder.
4. Alimentación (Vcc/GND) convencional.
5. Bus de conexión con el mando, con los pines de salida del esquema de la página 1.

En la fotografía de la placa se puede observar directamente la adaptación efectuada, que consiste en reemplazar los cables originales de los puertos (3), (4) y (5) por nuestras propias conexiones, que hemos llevado hacia el exterior. Asimismo, hemos mantenido sin modificar el montaje de los puertos (1) y (2).

A continuación, detallamos los cambios efectuados para conseguir el esquema de modificaciones propuestas para un motor, y extenderlo a todos los motores del manipulador:

1. Hacemos accesible el terminal (3) = “GND” = M-, mediante el cable blanco de la imagen. Dado que los terminales M- de todos los motores son el mismo (están conectados entre sí) este nuevo cable irá al terminal M- de todos los L298N.

2. Sustituimos la fuente doble original por una única fuente de 12V, conectando sus terminales al puerto (4). De esta forma, conectaremos los terminales (+) y (-) del bus (5) a todos los terminales Vcc y GND de los L298N.

3. Dado que el cable múltiple conectado al bus (5) es personalizado y tenemos accesibles todos los pines, conectaremos los terminales del “1” al “5” del bus a sus correspondientes terminales “M+” del L298N que controle ese motor.

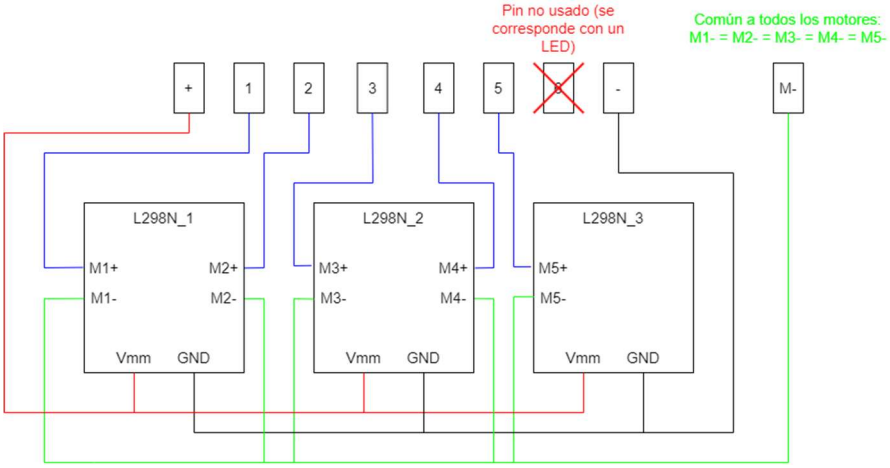


Figura 5. Conexionado de alimentación y control de los motores mediante controladores L298N (faltan los pines digitales)

Con estas modificaciones ya tenemos conectados los motores a drivers L298N, que además cuentan con su alimentación correspondiente. Faltan las señales digitales que gobiernan el comportamiento de los motores, y que decide el microcontrolador, que veremos más adelante.

NOTA: hemos sustituido la alimentación de 12V a 6V para los motores del brazo, gracias a un regulador L7806CV), dado que 12V era una tensión excesiva para los motores (iban muy rápido).

4.2. FUNCIONAMIENTO DE LA PLATAFORMA MÓVIL

4.2.1. DESCRIPCIÓN DEL SISTEMA

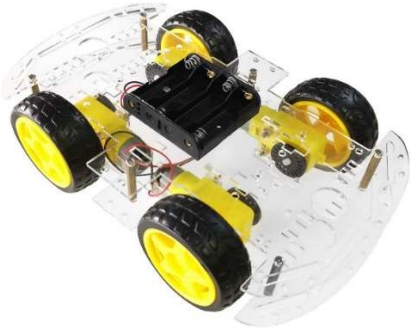


Figura 6. Plataforma móvil elegida para el prototipo, disponible en [2]

Ensamblaremos el manipulador del robot del paso 1 sobre esta estructura [2], que consta de dos plataformas, cada una de ellas equipada con cuatro ruedas y un motorreductor (compuesto por un motor y una caja reductora para regular la velocidad).

Se ha escogido un modelo que tenga suficiente potencia para soportar y mover el peso del manipulador, de la batería y de los componentes de interconexión.

Cada uno de los motorreductores tiene disponibles los dos pines del motor, M+ y M-, de tal forma que podemos controlar la velocidad de los motores a través de la tensión suministrada (por el controlador L298N).

4.2.2. MOTORES EN SERIE

En adelante, identificaremos la parte delantera del prototipo como aquella donde va ensamblado el manipulador, para identificar los motores izquierdos y derechos.

Tras sucesivas pruebas, hemos decidido cablear las ruedas de ambos lados del robot en serie, tratándolas como un único motor, pues ello conlleva una serie de ventajas:

1. Respecto a los drivers L298N, es importante señalar que en realidad cuentan con 2 terminales en lugar de 4. Esta configuración nos permite reducir la cantidad de señales digitales necesarias para controlar los motores. En consecuencia, simplificamos tanto el cableado como el control (y el código) de la plataforma móvil. Anteriormente requeríamos 3 pines por cada motor, pero al emparejar las ruedas, ahora solo utilizamos 6 pines en total, reduciendo significativamente la complejidad.

2. Otra ventaja de esta configuración es que facilita el mantenimiento de la dirección y los giros del robot. Las dos ruedas de ambos lados giran con la misma velocidad y referencia de tensión, por lo que se mantiene mejor la consigna de velocidad para cada lateral.

3. Si por cualquier casualidad una rueda queda bloqueada, la otra de su mismo lado la ayudará a girar nuevamente.

Figura 7. Identificación de los motores izquierdos y derechos del prototipo

Respecto a la alimentación, ahora la tensión se distribuye entre los dos motores de cada lado, por lo que hemos elegido una batería de 12V para alimentar a todos los motores.

4.2.3. ESQUEMÁTICO Y DETALLE DEL CABLEADO DE LA PLATAFORMA MÓVIL

El cableado en este apartado tiene como objetivos hacer llegar los 12 voltios a cada uno de los drivers encargados de los motores, conectar los terminales positivo y negativo de cada pareja de motores en el terminal M+ y M- del driver correspondiente a dicha pareja, y conectar las señales digitales al microcontrolador.

Aunque se podría haber cableado toda la plataforma móvil empleando un solo L298N (se tienen dos parejas de motores y cada controlador tiene dos puertos M+/M-), hemos decidido usar dos L298N debido a la separación lateral entre las parejas, y situar cada uno convenientemente cerca de sus respectivos motores.

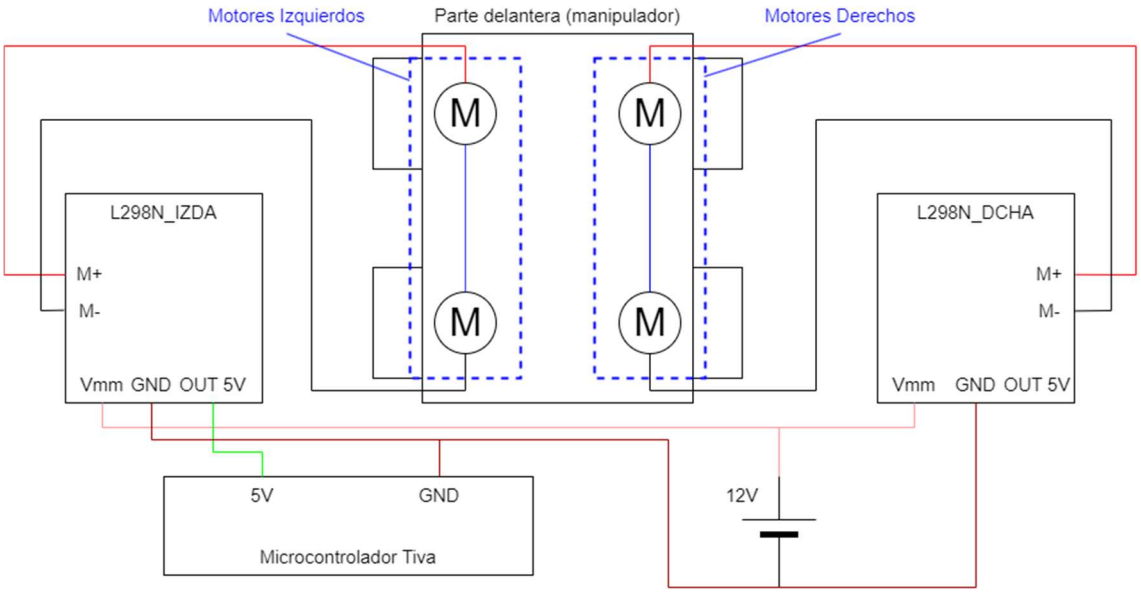


Figura 8. Conexionado de alimentación de los motores de la plataforma móvil. Aprovechamos que el L298N contiene un regulador de la tensión de entrada Vmm que emplea para mover los motores, con salida a 5V, para alimentar al microcontrolador.

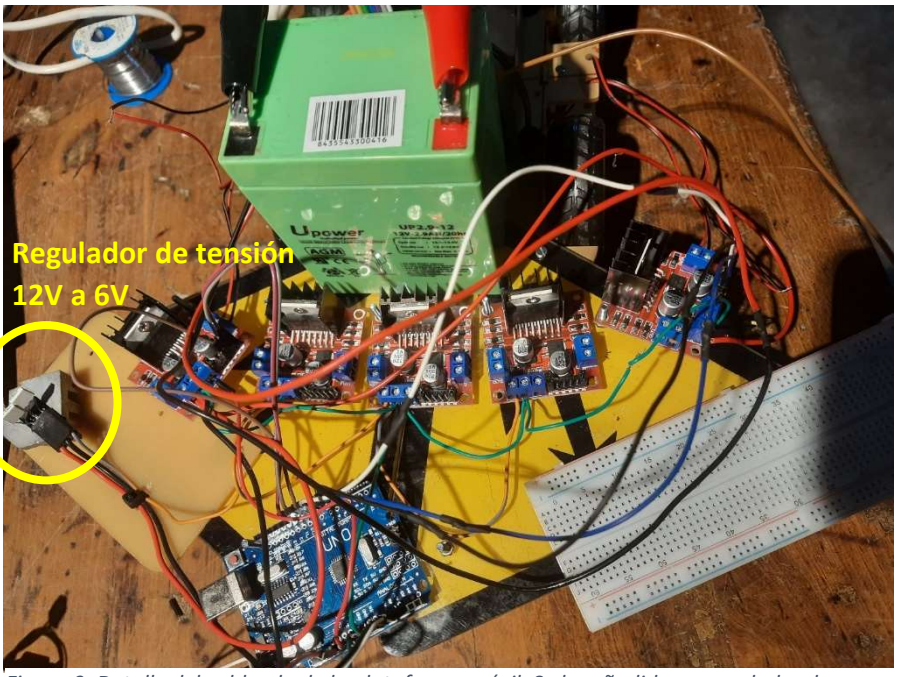


Figura 9. Detalle del cableado de la plataforma móvil. Se ha añadido un regulador de tensión de 12 a 6V, cuyo uso justificaremos más adelante.

4.3. MECANIZADO, CABLEADO Y ALIMENTACIÓN. CABLEADO COMPLETO DEL ROBOT (MANIPULADOR + PLATAFORMA MÓVIL) CON EL TIVA

En este apartado comentaremos algunos aspectos prácticos y algunas decisiones que se han tomado a la hora de construir, cablear y alimentar el robot.

4.3.1. MECANIZADO

Comentamos a continuación algunos aspectos prácticos del montaje de la estructura principal del robot.

1. El brazo y el coche son dos subsistemas independientes. Para unirlos, hemos atornillado los soportes del robot a la plataforma superior del coche.

2. Era necesario incluir en algún lugar el microcontrolador, los drivers de los motores, cables, batería y otros componentes. Dado que tras la unión no había mucho espacio (y muchos de los componentes no cabían ni por separado entre las dos plataformas del coche), se ha decidido colocar una bandeja en la parte trasera.

   Dado que esta bandeja va a contener la totalidad de los componentes electrónicos, incluida además la batería, de gran peso, hemos optado por situar el brazo en la parte delantera de la plataforma móvil para compensar así su peso y evitar que el robot pueda volcar.

3. Para situar los componentes electrónicos tales como los drivers o las tarjetas de desarrollo de los microcontroladores, hemos situado unos tornillos de un calibre tal que queden sujetos a la bandeja mediante unos agujeros presentes en su placa. No es una unión definitiva: el micro queda sujeta a la plataforma mediante el tornillo, pero puede extraerse con facilidad para permitir retocar algunas conexiones y/o programas.

Figura 10. Los dos pesos principales del robot: la parte delantera debido al manipulador, y la trasera a los componentes electrónicos y la batería

Figura 11. Tornillo de fijación del manipulador a la plataforma móvil

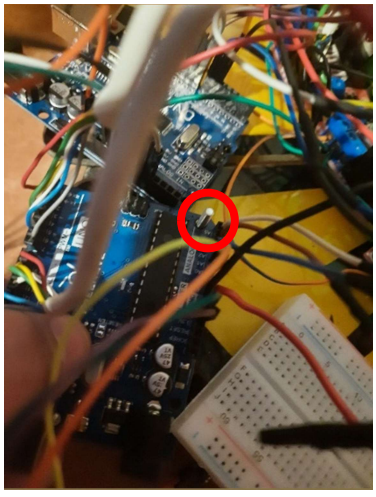


Figura 12. Tornillo para sujetar tarjetas, como los drivers o microcontroladores

4.3.2. CABLEADO

En esta sección, vamos a comentar el esquemático de alimentación completo del robot (obviando las señales digitales, que comentaremos más adelante):

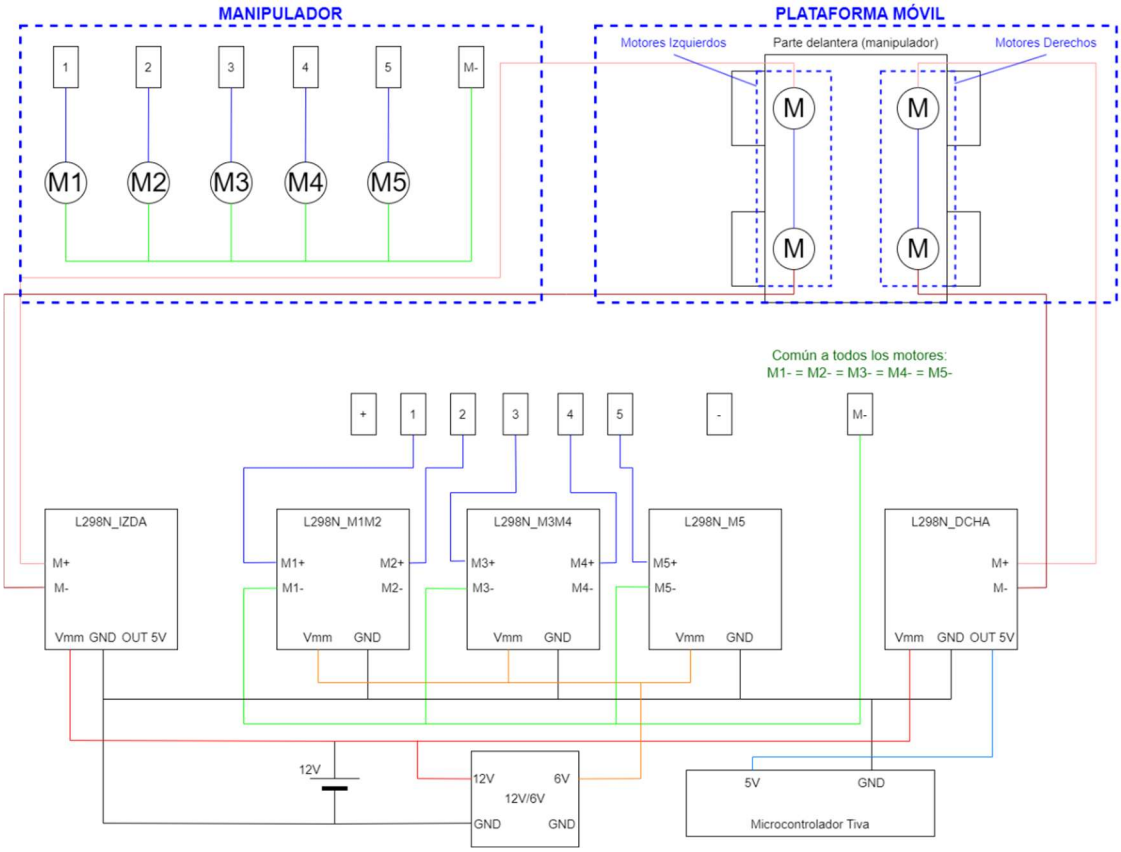


Figura 13. Conexionado de alimentación completo del robot. Los motores del manipulador se alimentan a 6V, y los de la plataforma móvil a 12V

En este diagrama, hay que tener en cuenta cuatro detalles:

1. Los GND de todos los dispositivos deben conectarse juntos y al terminal negativo de la fuente de 12V. De esta manera, no dejamos ningún elemento flotante (no se daña ningún circuito).

2. Los microcontroladores necesitan una alimentación de 5V. En principio, todos los L298N tienen una salida (si el jumper de regulación está conectado) de +5V, sin embargo, hemos comprobado que los L298N alimentados a 6V dan una tensión en esa salida de 3.7V aprox. Los de 12V, los de la plataforma móvil, son los que sí dan una tensión de +5V, por tanto conectamos ahí los microcontroladores.

3. Como vimos anteriormente, debemos modificar la PCB del brazo para poder acceder al terminal M-. Dado que a los drivers les hemos dado tensión a través de la batería externa, los terminales + y – del bus del manipulador ya no se usan, y por tanto no se han incluido en este diagrama. Asimismo, dado que el terminal 6 se corresponde con un led del brazo que no vamos a usar, tampoco usamos este terminal del bus.

4. Las pruebas iniciales se hicieron con dos Arduino UNO. En el montaje final, las hemos sustituido por el Tiva. La alimentación del TIVA se está realizando a través de la salida de 5V que proporciona uno de los L298N conectado a 12V (los de la plataforma móvil).

5. Mientras los reguladores de la plataforma móvil se alimentan a 12V, los motores del manipulador originalmente se alimentaban a una tensión inferior (3V). Si contamos con una caída de 3V en los L298N que los controlan, finalmente hemos elegido una tensión de 6V para los motores del manipulador, que se consiguen mediante un regulador de tensión, que comentaremos más adelante.

El cableado real del esquema anterior es el siguiente:

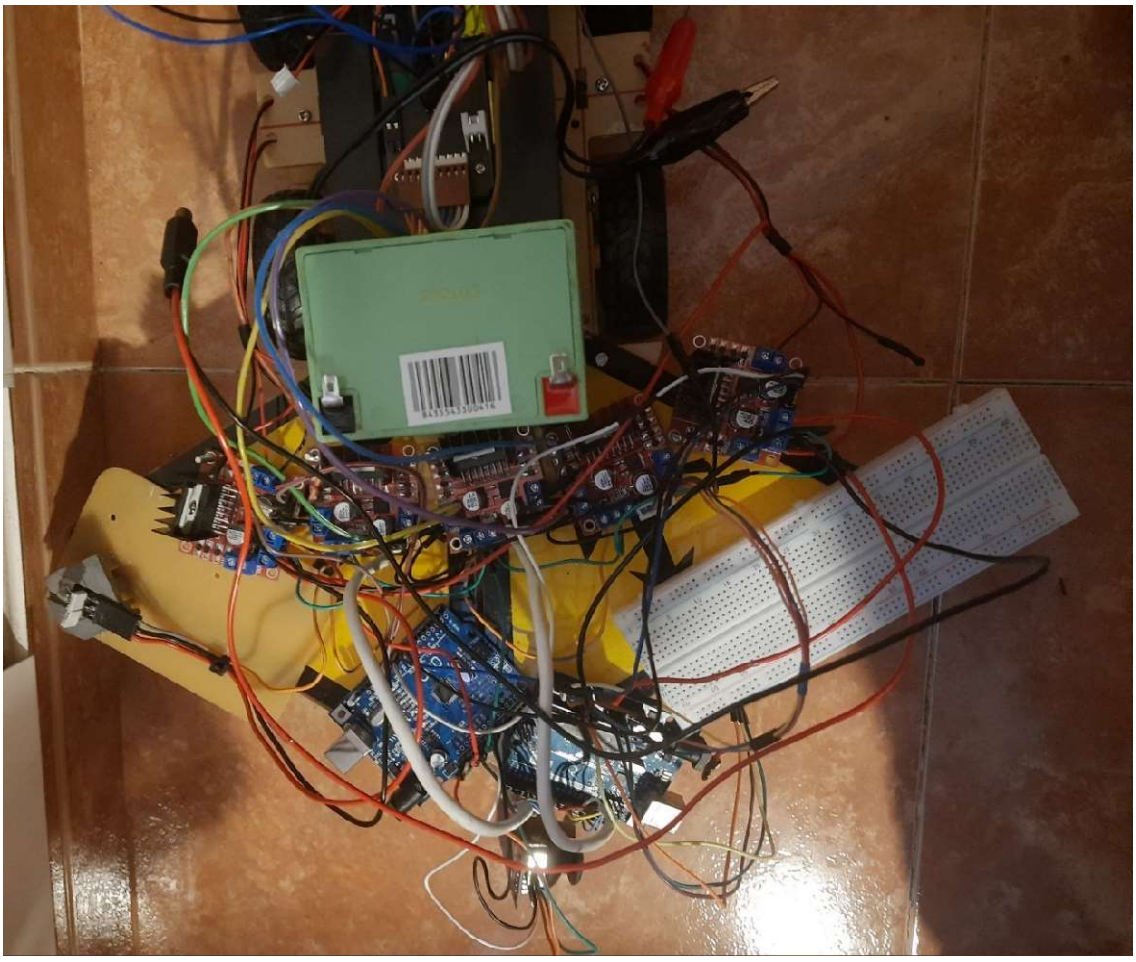


Figura 14. Detalle de conexiones del prototipo final. En la imagen se muestran dos Arduino UNO que se emplearon durante las pruebas de los motores, y que finalmente fueron sustituidos por el Tiva

Dada la complejidad de las interconexiones, en ocasiones, se presentaban situaciones en las que algunos cables se soltaban durante los movimientos del robot, lo que resultaba en un comportamiento inestable e impredecible del mismo. Para abordar esta cuestión, se tomaron medidas para mejorar la estabilidad y confiabilidad del cableado. Se optó por sustituir los cables más cortos y tensos por otros más largos, se reemplazaron las puntas de hilo por puntas rígidas o puntas de cables adecuadas para conectar a una protoboard, incluso se optó por soldar directamente algunas conexiones para evitar que se desprendieran.

Cuando fue necesario conectar varios cables en la misma regleta de entrada de los pines de uno de los drivers L298N, se procedió a realizar empalmes y soldaduras, permitiendo que el tornillo de la regleta pudiera apretar y asegurar adecuadamente dichas uniones. Por su parte, para evitar posibles tensiones y roturas en los cables del manipulador durante ciertos movimientos, se les proporcionó un cierto grado de libertad para asegurar su funcionamiento sin contratiempos. Con estas mejoras, el robot logró una mayor estabilidad y eficiencia en sus movimientos.

4.3.3. ENERGÍA

Inicialmente se usaron pilas de 9V para alimentar tanto el manipulador como la plataforma, sin embargo, tras sucesivas pruebas, se hizo evidente que los motores en serie de la plataforma necesitaban más tensión, y que los motores del manipulador, por el contrario, realizaban movimientos demasiado forzados, y requerían de una tensión inferior. Por otro lado, estas pilas se agotaban rápidamente tras unas cuantas pruebas, dado el enorme consumo de todos los motores juntos (5 del manipulador + 4 de la plataforma).

Para alimentar el robot se ha optado por usar una batería externa recargable de plomo de 12V, con una capacidad de 2900 mAh, suficiente para hacer todas las pruebas que se deseen sin necesidad de reponer en poco tiempo la carga.

Los drivers tienen durante sus operaciones de control una caída de tensión de 3V. Por tanto, los motores perciben la tensión de entrada del driver – 3V. Los motores de la plataforma se conectan a 12V a través del driver, por tanto perciben una tensión efectiva de 9V que deben repartirse entre los dos motores de cada lateral (izquierdo y derecho), dado que están en serie.

Por otro lado, los motores del manipulador se están conectando a 6V a través de sus correspondientes drivers. Por tanto, están percibiendo una tensión efectiva de 3V, justo la tensión máxima con la que operaban en el diseño original (las baterías originales se han conservado para aprovechar su peso).

Para convertir la tensión de 12V a 6V, se ha usado el circuito integrado L7806CVy lo hemos configurado con dos condensadores en paralelo. Para que funcione como regulador de tensión a 6V, es necesario conectarlo en dicha configuración, que mostramos a continuación, con dos condensadores. Dado que es un integrado propenso a calentarse, le hemos conectado un disipador de hierro.

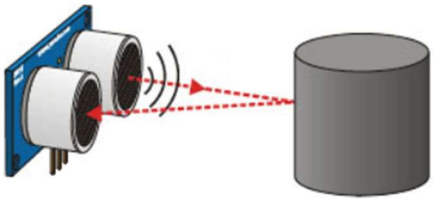
Figura 15. Conexión final del integrado con los condensadores (parte superior) y el disipador de hierro

4.3.4. EL MÓDULO ULTRASONIDOS HC-SR04

Hemos añadido en la parte frontal del robot un módulo ultrasonidos, cuya finalidad es bloquear el avance del robot cuando se encuentre frente a un obstáculo para evitar un posible impacto. Con este sensor, medimos el tiempo entre el envío y la recepción de un pulso de ultrasonidos. Si tenemos en cuenta que la velocidad del sonido es 343 m/s, dado que vamos a contar el tiempo en µs, la convertimos a cm/µs. Por otro lado, tenemos que tener en cuenta que realmente el sensor va a medir el tiempo desde que el pulso de ultrasonidos se emite hasta que vuelve. Por tanto, la distancia final (tiempo/velocidad) debe dividirse por la mitad para dar la distancia real del objeto [3].

$$343\frac{m}{s}\cdot 100\frac{cm}{m}\cdot\frac{1}{1000000}\frac{s}{\mu s}=\frac{1}{29.2}\frac{cm}{\mu s}$$

$$Distancia(cm)=\frac{Tiempo(\mu s)}{29.2\cdot 2}$$



Tiempo = 2 * (Distancia / Velocidad)
Distancia = Tiempo · Velocidad / 2

Figura 16. Cálculo de la distancia en centímetros empleando la velocidad del sonido en cm/µs = 1/29.2. El resultado se divide por dos debido a que se ha contabilizado el tiempo que tarda el pulso sonoro en ir y en volver. Fuente: [3]

NOTA: hemos aproximado 1/29.2x2 como 1/59, de tal manera que la distancia se calcula dividiendo el tiempo de vuelo medido por 59.

Cabe destacar que este módulo se debe alimentar necesariamente a 5V, y que las señales que emita de vuelta al micro van a ser de 5V. Por su parte, los pines GPIO del Tiva admiten una tensión máxima de 3.3V (tanto para entrada como para salida), por lo que es necesario hacer una adaptación de tensiones.

Para realizar dicha adaptación, hemos optado por el uso de un módulo Level Shifter. Se trata de un componente que permite adaptar señales digitales, escalándolas de un nivel de tensión a otra o viceversa, de forma totalmente bidireccional [4].

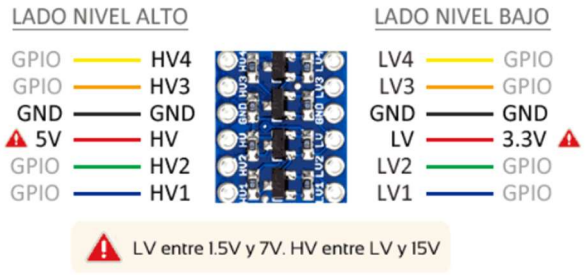


Figura 17. Ejemplo de uso del módulo Level Shifter. Fuente: [4]

Este módulo presenta dos interfaces, una de alta tensión o HV, y otra de baja tensión o LV. En el lado HV alimentaremos a 5V, y en el lado LV a 3.3V. Por otro lado, conectaremos la GND común a todos los dispositivos del robot, a GND de HV y de LV (la misma GND en HV y en LV, que debe ser además la GND común de todo).

Con este cableado, las señales digitales que se conecten ahora en los terminales HVi a LVi y viceversa se escalarán bidireccionalmente: si tenemos una señal digital de 5V en HV2, en LV2 tendremos una señal digital con la misma amplitud y desfase, pero escalada a 3.3V. Funciona exactamente igual al revés. De esta forma, estamos conectando los pines del ultrasonidos a señales de 5V, y los puertos GPIO del Tiva a señales de 3.3V.

A continuación, mostramos algunas imágenes del montaje final del ultrasonidos:

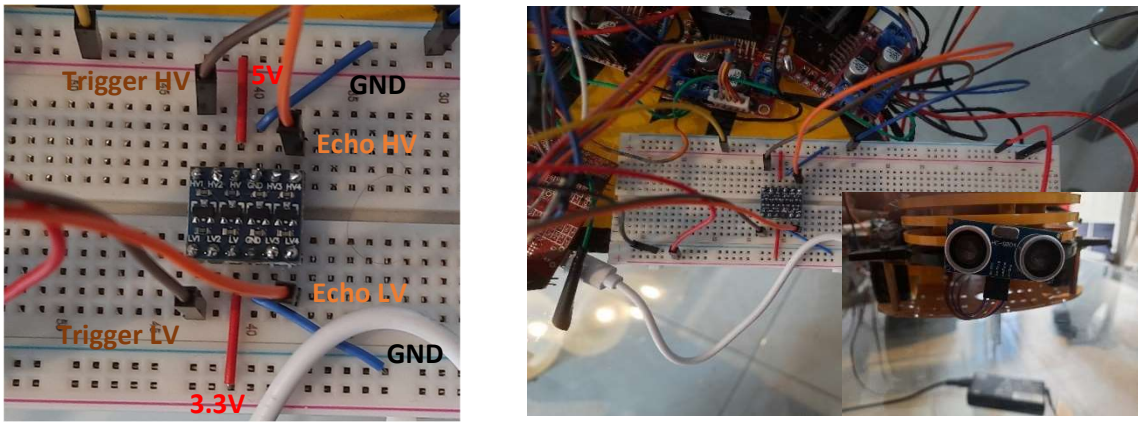


Figura 18. Detalle del conexionado del módulo Level Shifter, ubicado en la parte trasera del prototipo. Por su parte, el sensor ultrasonidos se ha situado en la parte delantera

4.3.5. CABLEADO GENERAL DE SEÑALES DIGITALES DEL ROBOT

Mostramos a continuación el cableado de todas las señales digitales del robot con el Tiva, que contiene todos los módulos y elementos comentados anteriormente:

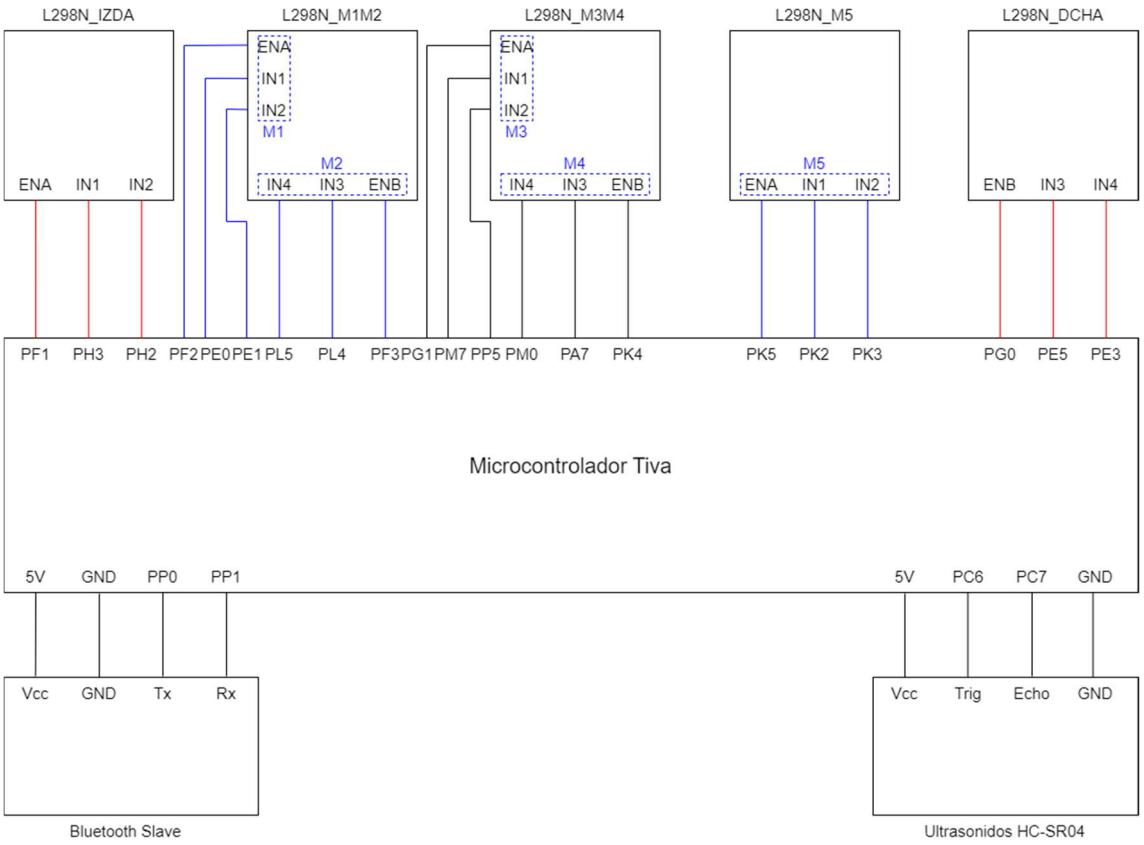


Figura 19. Conexionado de todas las señales digitales empleadas en el microcontrolador con los pines de los diferentes módulos empleados. Aunque el módulo Bluetooth se alimenta a más de 3.6V, sus pines Tx y Rx pueden trabajar a 5V, por tanto no se necesita Level Shifter. Sí lo necesitan las señales Trig y Echo (5V) al conectarse a los pines digitales del microcontrolador (3.3V), pero no se ha añadido en el esquema por simplicidad.

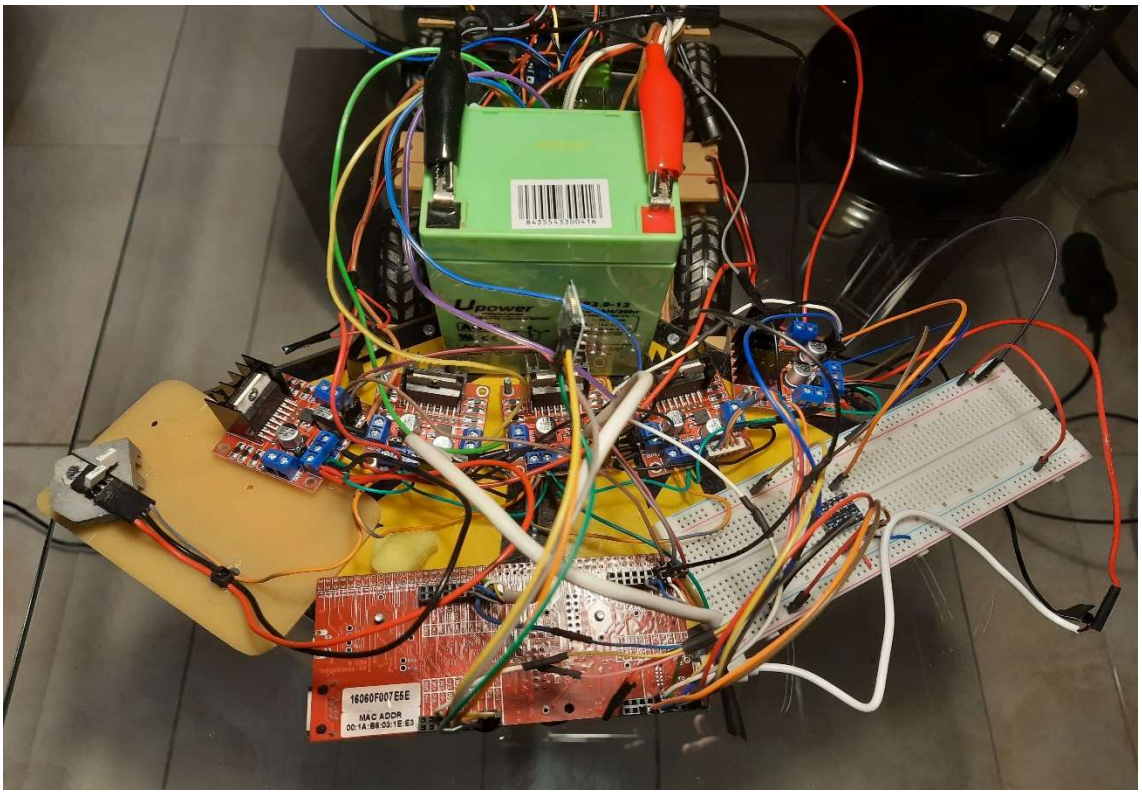


Figura 20. Cableado completo del robot, con todas las señales digitales, el regulador a 6V, el ultrasonidos, el Level Shifter y el microcontrolador Tiva

4.4. CABLEADO DEL MANDO BLUETOOTH

Una vez resuelta la estructura hardware del robot, pasamos a la del mando de control del mismo. En el caso del mando, hemos conectado a su correspondiente microcontrolador Tiva dos joystick y el módulo Bluetooth que actuará de master de la comunicación. También hemos conectado la pantalla con su placa de adaptación al Boosterpack 2 del micro, y el resto de elementos a los pines del Boosterpack 1:

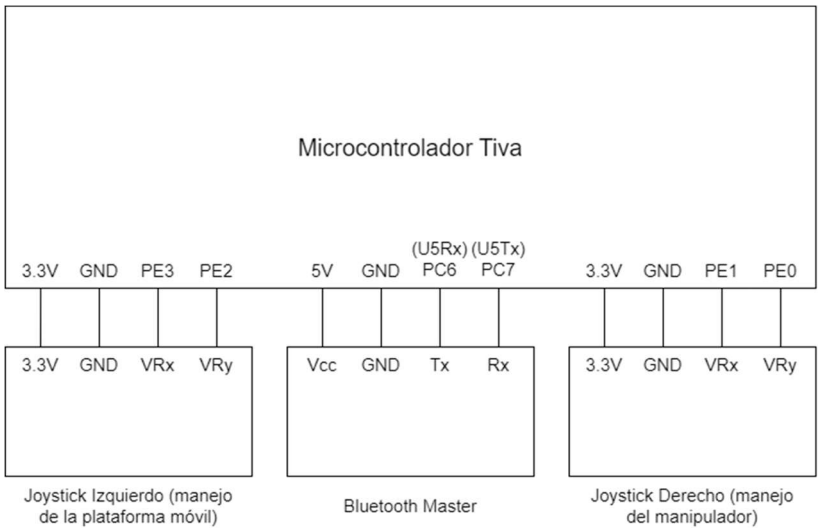


Figura 21. Cableado del mando de control del prototipo. Aunque el joystick está pensado para 5V, también funciona completamente a 3.3V, sin necesidad de adaptar tensiones. Por su parte, como ya hemos comentado, el módulo Bluetooth se alimenta a 5V, pero sus pines Tx y Rx pueden trabajar perfectamente a 3.3V

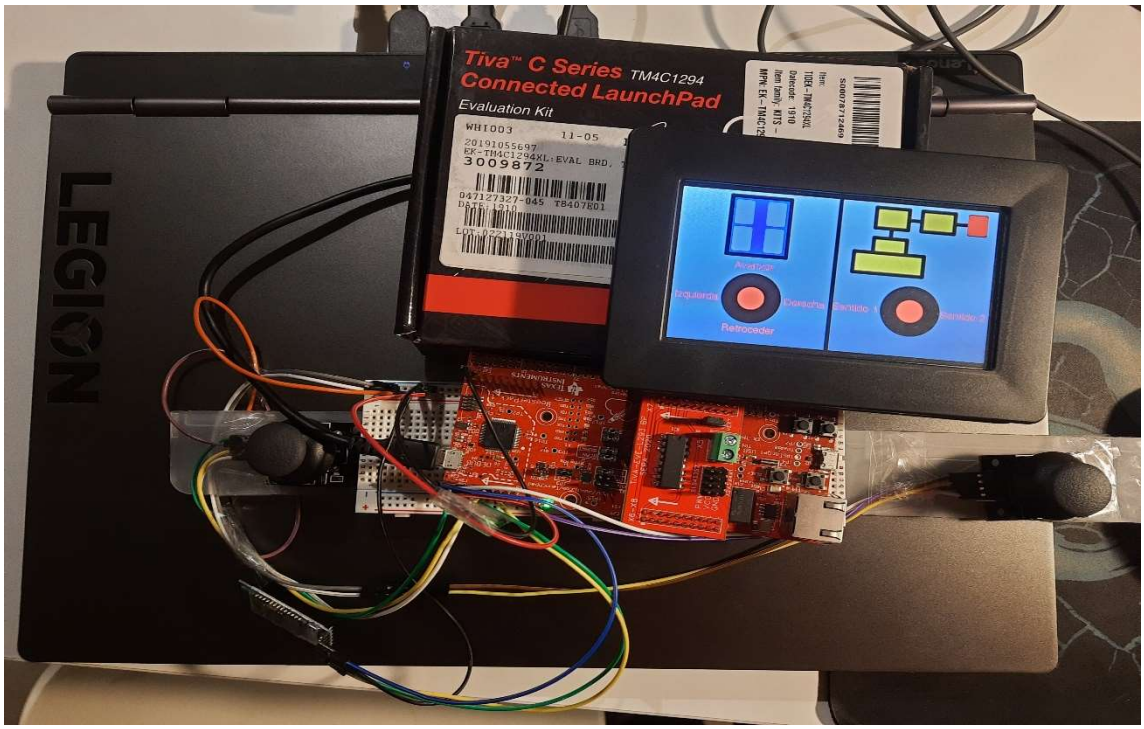


Figura 22. Resultado final del mando de control del prototipo. Cuenta con dos joystick, un módulo Bluetooth y una pantalla táctil

Los pines de los joystick se han conectado a pines del micro con funcionalidad de conversión ADC. El Tiva tiene dos ADC, compartidos entre los 20 pines de entrada analógica posibles del microcontrolador. En nuestro código, usaremos el ADC0.

5. DESARROLLO SOFTWARE DEL PROYECTO

5.1. SOFTWARE DEL ROBOT

5.1.1. VARIABLES EMPLEADAS EN EL CÓDIGO DEL PROTOTIPO

VARIABLES DE LA PLATAFORMA	DESCRIPCIÓN
VelCoche	Variable que almacena el periodo PWM para configurar la velocidad de la plataforma
EjeXDcha	Almacena valor eje x joystick derecho
EjeYDcha	Almacena valor eje y joystick derecho

VARIABLES DEL MANIPULADOR	DESCRIPCIÓN
VelBrazo	Variable que almacena el periodo PWM para configurar la velocidad de la articulación elegida del manipulador
EjeXIzda	Almacena valor eje x joystick izquierda
EjeYIzda	Almacena valor eje y joystick izquierda
MotorElegido	Valor que almacena la articulación del manipulador que se va a mover con el joystick

VARIABLES UART	DESCRIPCIÓN
UARTflag	Flag de interrupción que se activa cuando se recibe una transferencia de datos completa del mando (cada ciclo incluye las lecturas de los 2 ejes de cada joystick, y la articulación del brazo elegida para moverse con el joystick del brazo)
buffer	Buffer para almacenar una transmisión completa del mando. El mando envía en este orden: Carácter de control, EjeXIzda8bitsMásSig, EjeXIzda8bitsMenosSig, EjeYIzda8bitsMásSig, EjeYIzda8bitsMenosSig, EjeXDcha8bitsMásSig,EjeXDcha8bitsMenosSig, EjeYDcha8bitsMásSig,EjeYDcha8bitsMenosSig, MotorElegido. El carácter de control ('A') sirve para sincronizar la emisión y la recepción.
i	Variable para recorrer el buffer de recepción de la comunicación serie. Debe estar fuera de la rutina de interrupción para mantener el valor entre distintas recepciones (interrupciones).

VARIABLES ULTRASONIDOS	DESCRIPCIÓN
d	Distancia en cm medida por el ultrasonido
t	Tiempo en microsegundos que cuenta el Timer1
t1	Tiempo en microsegundos que tarda el echo en pasar de 0 a 1 (medir el ancho de echo)
t2	Tiempo en microsegundos que tarda el echo en pasar de 1 a 0 (medir el ancho de echo)
flag	Flag de interrupción para la rutina de interrupción del ultrasonido

VARIABLES MAPEO JOYSTICK	DESCRIPCIÓN
Max_vel	Valor máximo del PWM determinado experimentalmente para que la velocidad de la plataforma sea máxima
Min_vel	Valor mínimo del PWM de tal manera que la velocidad de la plataforma es mínima pero no nula (determinada experimentalmente)
mMenos	Pendientes de las rectas de mapeos de velocidades de plataforma y manipulador eje joystick de 1850 a 0
mMas	Pendientes de las rectas de mapeos de velocidades de plataforma y manipulador eje joystick de 2300 a 4096
bMenos	Ordenadas en el origen de las rectas de mapeos de velocidades de plataforma y manipulador de 1850 a 0
bMas	Ordenadas en el origen de las rectas de mapeos de velocidades de plataforma y manipulador de 2300 a 4096

5.1.2. DIAGRAMAS DE FLUJO DEL CÓDIGO

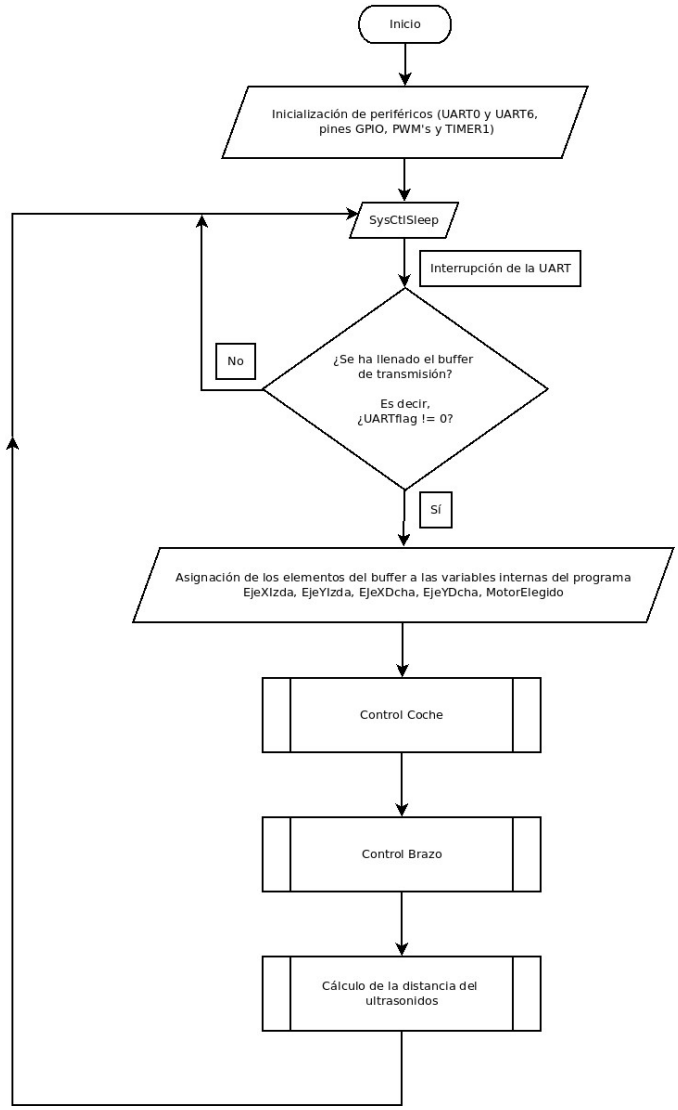


Figura 23. Diagrama de flujo general del código del robot

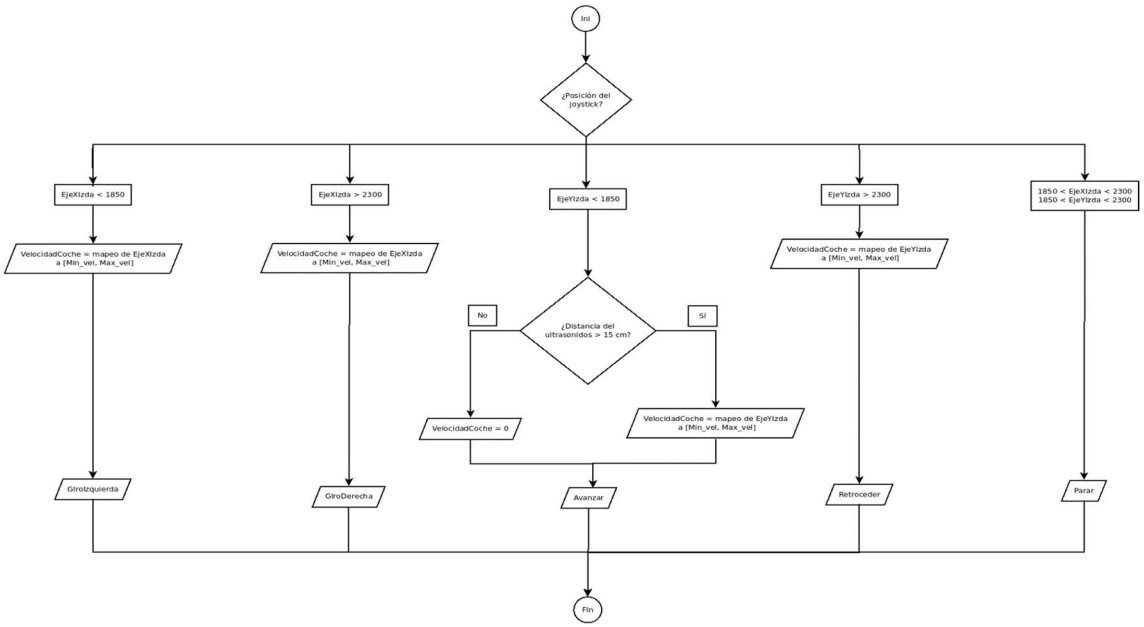


Figura 24. Subrutina "Control Coche"

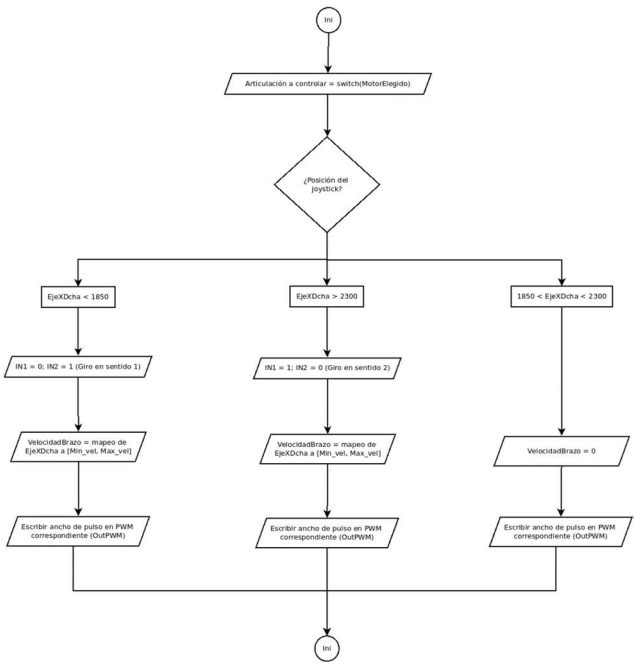


Figura 25. Subrutina "Control Brazo"

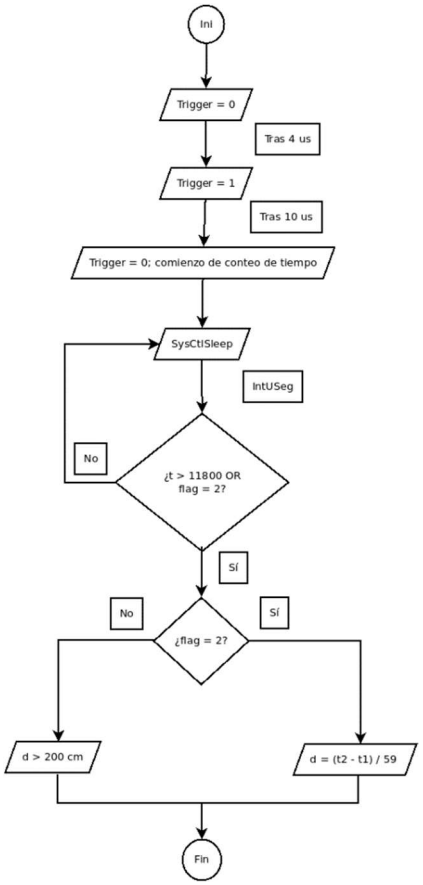


Figura 26. Subrutina "Cálculo Distancia Ultrasonidos"

5.1.3. DESCRIPCIÓN DEL CÓDIGO DEL ROBOT

El código encargado del control del robot recibe el nombre de **RobotFinal.c**. Adentrándonos en el main, iniciamos el programa llevando a cabo la configuración de todos los puertos GPIO. Además, se configuran 2 UART:

- Por un lado, la UART0 se emplea para la monitorización de variables a través de consola a modo de depuración (siempre y cuando esté conectado a un puerto USB de un PC). Esta UART se configura haciendo uso de la librería **uartstudio.c**.

- Por otro lado, configuramos la UART6 (PP0=>RX; PP1=>TX) manualmente para la recepción de datos vía Bluetooth. Las dos UART se configuran a 115200 baudios.

Acto seguido, configuramos los 7 PWM necesarios para el control de los motores del robot (todos los PWM disponibles). Esta configuración se realiza con la función **configuraPWM50Hz** desarrollada en la librería propia **LibreriaRobotSEPA.c.** Se establece la configuración de los pines entrada-salida correspondientes a los motores y ultrasonidos. Por último, configuramos el TIMER1A como timeout, periódico con periodo de 1us. También hemos configurado una serie de interrupciones:

- Una para la UART6 (**Int UART**) para la recepción de datos. Interrumpirá al micro cada vez que llegue un dato discreto, no una transmisión completa (consistente en los siguiente datos que envía el mando: EjeXIzda, EjeYIzda, EjeXDcha, EjeYDcha, MotorElegido) e irá así llenando la variable **buffer**, un buffer de recepción. Sólo levantará su flag cuando se llene dicho **buffer**.

- Una para el pin "echo" del ultrasonido configurada para activarse por flanco de subida y bajada (**rutina_interrupcion_echo**) y que nos ayudará en la medida de la distancia.

- Una para el timer1(**IntUSeg**), que llevará una cuenta del tiempo en microsegundos. Desactivamos temporalmente la interrupción del pin echo para evitar que este salte inesperadamente e iniciamos la rutina de control.

En el bucle infinito, lo 1º que realiza el programa es pasar a modo sleep a la espera de recibir un dato por la UART6, al recibir un dato, salta la rutina de interrupción **Int UART,** en la que vamos montando el mensaje, que se va almacenando en la variable **buffer.** El mensaje se almacenará en el orden correcto gracias a la existencia del carácter de control que se envía desde el mando. Dicho carácter es la letra ‘A’.

Sólo cuando se recibe un mensaje completo y se llene el buffer, la rutina de interrupción activa el flag **UARTflag**, saliendo del modo sleep y asignando a variables internas del programa del robot (variables EjeXIzda, EjeYIzda, EjeXDcha, EjeYDcha y MotorElegido) el valor del eje x e y de cada joystick y el motor elegido. Cabe destacar que debido a que el dato enviado por UART es de 8 bits y el valor de los ejes de los joysticks proviene del ADC del micro de 16 bits, ha sido necesario descomponer los valores de los ejes: se envían primero los 8 bits más significativos del valor leído por el ADC y justo a continuación, los 8 bits menos significativos.

Tras recibir y almacenar la secuencia de comunicación Bluetooth, es decir, las lecturas de los joystick y la pantalla, el bucle infinito ejecuta los controles de la plataforma y del manipulador por separado, y posteriormente interpreta la lectura del ultrasonidos:

- Para el control de la plataforma, se hace uso de las variables **EjeXIzda** y **EjeYIzda** (es el joystick izquierdo el que maneja la plataforma móvil del robot). En caso de que el **EjeYIzda** sea inferior a 1850 (joystick hacia delante) y la distancia medida por el ultrasonido sea superior a 15 cm procedemos a calcular la velocidad de los motores, mapeando la lectura del joystick al rango [**Min_vel, Max_vel**], y avanzamos a esa velocidad. Para ello, usamos la función **Avanzar** de la librería **LibreriaRobotSEPA.c.**
  Por el contrario, en caso de que el eje Y sea mayor a 2300 calculamos la velocidad mapeando la lectura del joystick de la misma manera, y procedemos a retroceder haciendo uso de la función con el mismo nombre, también desarrollada en la librería propia.
  Por otro lado, el valor del eje X nos indicará el giro del robot, si es superior a 2300 girará a la derecha y si es inferior a 1850 a la izquierda, siempre con una velocidad proporcional al valor del joystick.
  Por último, en caso de que el joystick se encuentre en posición central, el robot permanecerá quieto. Las funciones de **GiroIzquierda, GiroDerecha y Parar**, que usamos en estos últimos casos, también se encuentran en la librería.

- Para el control del manipulador, en función de la variable **MotorElegido**, controlaremos una articulación u otra. Para ello, asignamos a las variables **PuertoIN1, PuertoIN2, PinIN1, PinIN2 y OutPWM** el valor del puerto y pin al que se encuentra conectados IN1, IN2 y PWM del motor correspondiente. Tras esto, en función la variable **EjeXDcha,** moveremos el motor correspondiente en un sentido u otro controlando la velocidad en función del valor del joystick derecho.

- Por último, llevamos a cabo la medida de la distancia del ultrasonidos, para lo cual ponemos a 0 el PIN_6 del puerto C ("trigger") durante 4us, recomendable para garantizar un nuevo disparo. Posteriormente, ponemos el pin a 1 durante 10us y lo volvemos a poner a 0, y, tras esto, esperamos a que se active la interrupción del pin echo, que medirá el ancho de pulso del echo recibido, detectando el tiempo que ha tardado "echo" en ponerse a 1 desde que se envió "trigger" (t1) y el tiempo desde que se envió "trigger" hasta que "echo" vuelve a ponerse a 0 (t2). Este ancho de pulso de "echo" se usará para el cálculo de la distancia a la que se encuentra el obstáculo que se encuentre delante del robot.
  En caso de que no se reciba el "echo" en 11800 us, la medida se considerará superior a 2m.

5.2. SOFTWARE DEL MANDO DE CONTROL

5.2.1. VARIABLES EMPLEADAS EN EL CÓDIGO DEL MANDO

VARIABLES MAPEO JOYSTICK	DESCRIPCIÓN
mIzdaX, mIzdaY, mDchaX, mDchaY, bIzdaX, bIzdaY, bDchaX, bDchaY	Pendientes y ordenadas en el origen para mapear el valor del joystick a pixeles en la pantalla, será usado para pintar una bola roja que indicará la posición del joystick
MaxEje, MinEje;	Valor máximo y mínimo de los ejes del joystick, usada para el cálculo del mapeo
PixelX, PixelY	Posición final donde se dibuja el joystick en pantalla

VARIABLES QUE SE ENVIARAN POR UART	DESCRIPCIÓN
DatoXDcha	Valor leído del eje X derecho por el ADC
DatoYDcha	Valor leído del eje Y derecho por el ADC
DatoXIzda	Valor leído del eje X izquierdo por el ADC
DatoYIzda	Valor leído del eje Y izquierdo por el ADC
MotorElegido	Variable que almacena el valor del motor elegido

VARIABLES ADC	DESCRIPCIÓN
ADC0flag	Flag que se activa en la rutina de interrupción del ADC0
pui32ADC0Value	Buffer donde se almacena el valor de los 4 canales del ADC0

5.2.2. DIAGRAMAS DE FLUJO DEL CÓDIGO

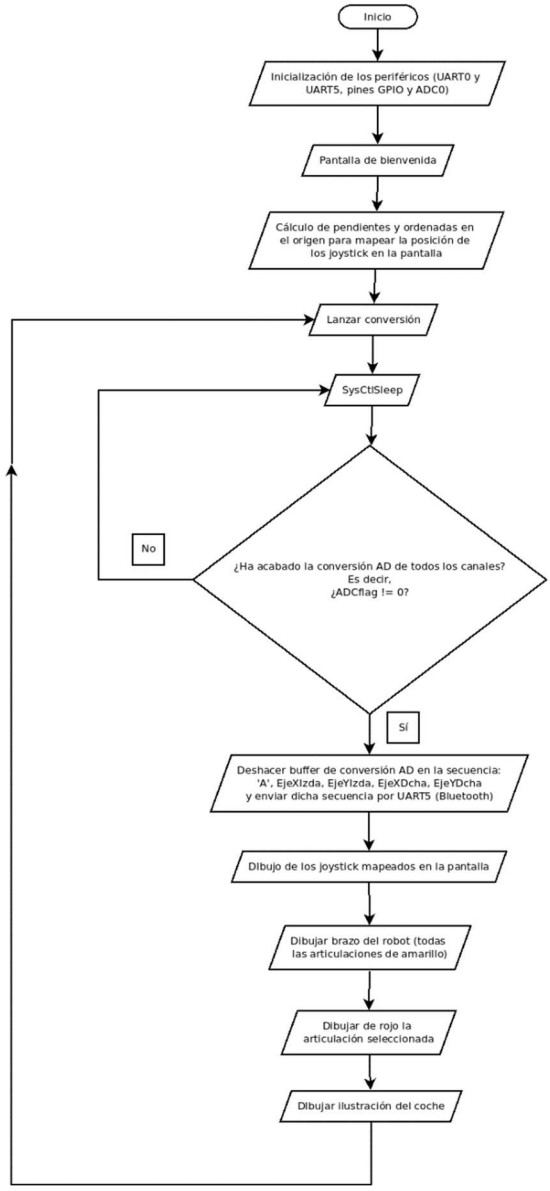


Figura 27. Diagrama de flujo general del software de control del mando

5.2.3. DESCRIPCIÓN DEL CÓDIGO DEL MANDO

En el código **MandoBTFinal.c** se desarrolla la funcionalidad del mando Bluetooth. Para comenzar, llevamos a cabo la configuración de los distintos elementos que usará nuestro mando. Por un lado, contamos con la UART0 configurada a través de la librería **uartstudio.c**, usada para monitorizar por PuTTY los datos enviados por Bluetooth. Por otro lado, configuramos la UART5 (PC6=>RX, PC7=>TX) de manera manual, para llevar a cabo la comunicación Bluetooth con el robot.

Acto seguido, configuramos el ADC0 (periférico no usado en las prácticas de la asignatura). Lo configuramos de tal forma que realice la conversión de 4 canales, el canal **AIN0** asignado al pin **PE3** (eje X del joystick izquierdo), el canal **AIN1** asignado al pin **PE2** (eje Y del joystick izquierdo), el canal **AIN2** asignado al pin **PE1** (eje X del joystick derecho) y el canal **AIN3** asignado al pin **PE0** (eje Y del joystick derecho). Se habilita el **ADC0** y limpiamos el posible flag de interrupción activo. A continuación, asignamos a este mismo convertidor la interrupción **InterrADC0,** de tal forma que mientras no realice una conversión completa el micro se quede en estado de reposo.

Tras finalizar la configuración inicial, mostramos la pantalla inicial y esperamos a una pulsación en la misma para iniciar el funcionamiento normal del mando. Asignamos valores a las distintas variables que nos ayudarán con el mapeo de los joysticks. En concreto, valores de píxeles en la pantalla (pendientes y ordenadas en el origen). Tras esto, comienza el bucle infinito, en el que encontramos las siguientes partes:

- Activamos la conversión del ADC0 y ponemos el micro en modo sleep, de tal forma que, hasta que no finalice la conversión de los 4 canales, permaneceremos en este estado.

- Una vez hayamos recibido todas las lecturas de los joystick, enviamos por Bluetooth (UART5) los datos leídos:

    - Almacenamos el valor de la conversión de los canales del ADC0 en el buffer **pui32ADC0Value**.
    - Para iniciar la comunicación, enviamos por la UART5 un carácter ‘A’, a modo de carácter de control de inicio de la comunicación.
    - Descomponemos el buffer **pui32ADC0Value**: almacenamos el valor del canal 0 en **DatoXIzda** y enviamos por la UART5 (Bluetooth) los 8 bits más significativos seguidos de los 8 bits menos significativos de **DatoXIzda,** acto seguido hacemos lo mismo con el canal 1 (**DatoYIzda**), el canal 2 (**DatoXDcha**) y el canal 3 (**DatoYDcha**). Por último, enviamos la variable **MotorElegido.**
    - Además de enviarse los datos por comunicación Bluetooth, también se envían estos mismos por la UART0 para la monitorización del sistema a través de PuTTY.

- Seguidamente, se realiza el dibujo de la interfaz de usuario, que mostramos más adelante en el apartado "manual de usuario". Para colorear la articulación elegida de color rojo, se comprueba el valor de la variable **MotorElegido,** variable que almacena la articulación elegida del manipulador.

- Por último, comprobamos si se ha pulsado en pantalla alguna de las articulaciones del manipulador para elegirla como articulación a controlar.

5.3 CÓDIGO DE LA LIBRERÍA "LibreriaRobotSEPA.c"

En esta librería se desarrollan una serie de funciones dedicadas a la configuración de los PWM de nuestro microcontrolador a 50Hz, además del manejo del vehículo:

- La función **configuraPWM50Hz** recibe como parámetros de entrada el identificador del PWM a configurar, el identificador del puerto GPIO que este usa, la base del PWM, el identificador del pin PWM GPIO, la base del puerto del GPIO, el pin del GPIO al que se conecta el PWM, el generador del PWM, la salida del PWM y por último OutPWM_BIT. Con todos estos argumentos, configuramos el PWM correspondiente a 50 Hz.

- **Avanzar** establece el pin **PE5 a 0** y activa **PE3** (ruedas de la derecha avanzan)**,** el **PH2 a 0** y activa **PH3** (ruedas de la izquierda avanzan). Por último, damos el valor de **velMotorMapeada** a los PWM de las ruedas. **Retroceder** lleva a cabo la misma función, pero invirtiendo los pines correspondientes al lado derecho e izquierdo con respecto a **Avanzar**. Por tanto, esta función sirve para que el robot retroceda a la velocidad indicada.

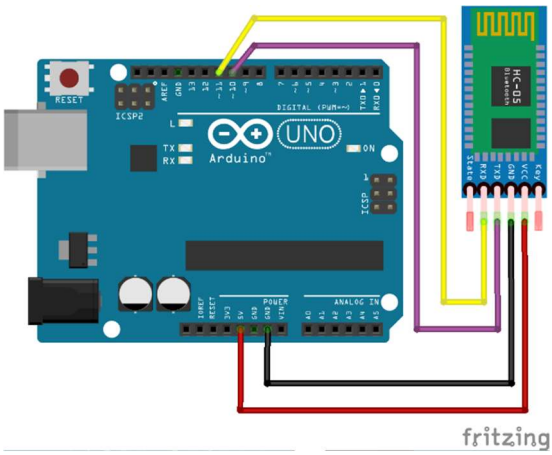
- Por último, **GirarDerecha** y **GirarIzquierda** se usan para el giro del robot. Al tratarse de un robot con configuración diferencial hacemos que las ruedas de la izquierda giren hacia delante y las de la derecha hacia atrás para el giro a la izquierda y a la inversa para el giro a derechas.

5.4. PROGRAMACIÓN DE LOS MÓDULOS BLUETOOTH (HC-05)

El módulo Bluetooth HC-05 es un módulo que puede ser programado como Master o Slave de la comunicación. Se conecta a la UART del microcontrolador, y se comunica por ella a través de su protocolo, mediante sus líneas Tx y Rx [5].

Para programar los módulos, los hemos configurado a la misma velocidad (115200 baudios), con una contraseña = 1212, y hemos configurado el módulo Bluetooth del mando como Master y el del robot como Slave. De esta manera, nada más conectarse, el Master busca al Slave, y cuando se encuentran, se establece la comunicación entre el Master y el Slave, que a su vez se comunican con la UART del microcontrolador al que estén conectados.

Para configurar los módulos Bluetooth, hemos realizado esta configuración usando un Arduino UNO y el terminal serie del Arduino IDE, con el siguiente montaje y el siguiente código de ejemplo:




```
#include <SoftwareSerial.h>

SoftwareSerial mySerial(10, 11); // RX, TX

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
  mySerial.begin(38400);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }


  Serial.println("Listo");
}

void loop() { // run over and over
  if (mySerial.available()) {
    Serial.write(mySerial.read());
  }
  if (Serial.available()) {
    mySerial.write(Serial.read());
  }
}
```

Figura 28. Conexionado y código de ejemplo para configurar los módulos Bluetooth

En el código anterior, la librería SoftwareSerial nos permite establecer los pines 10 y 11 como pines de UART (el Arduino no carga el código si hay algo cableado en sus pines 0 y 1, que son los pines de UART reales, ya que el código se carga mediante el puerto serie del micro).

Se configura el módulo a 38400 baudios, que es su velocidad de configuración, por la cual el módulo se establece en modo configuración. Abrimos el terminal serie tras cargar el programa.

Cabe destacar que se debe desconectar la alimentación del módulo, cargar el programa, y alimentar el módulo con el botón presente en el módulo pulsado. Esto, junto con el hecho de configurar la velocidad de la comunicación a 38400 baudios, hace que el módulo entre en el llamado modo AT o modo de configuración.

5.4.1. COMANDOS DE CONFIGURACIÓN DEL MÓDULO

Siguiendo los pasos descritos en [5], en primer lugar, comprobaremos en el terminal serie si el módulo se ha establecido correctamente en modo de configuración. El led del módulo debería parpadear de forma lenta en este modo. Si ejecutamos el comando AT, debería responder con un OK, lo que significa que el módulo ha entrado correctamente en modo de configuración y que está preparado para ser configurado:

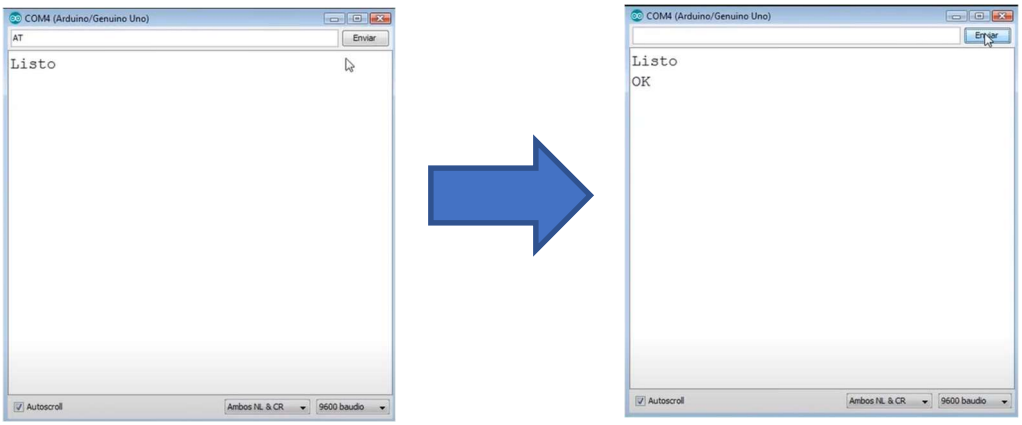


Figura 29. Se envía el comando AT por terminal serie y se comprueba si el módulo responde con "OK" [5]. La comunicación con el módulo se hace a 38400 baudios, mientras que el terminal está configurado a 9600 baudios, como se aprecia en la imagen

A partir de ahí, deberemos ejecutar los siguientes comandos (que si dan como respuesta un OK es que se han ejecutado y configurado correctamente) [5]:

```
AT + UART = 115200,0,0 //Comunicación a 115200, sin bits de paridad ni stop
AT + PSWD = “1212”     //Configuramos contraseña entre slave y master
```

A partir de aquí, ejecutaremos comandos distintos en el Master y en el Slave [5]:

- Slave:

  ```
  AT + ROLE = 0   // ROLE = 0 => Slave;   ROLE = 1 => Master
  AT + ADDR?      // Devuelve por pantalla su dirección MAC, que tendremos
                  // que introducir en el Master
  ```

  Copiaremos esta dirección MAC en un bloc de notas, y para poder introducirla en el Master, cambiaremos los “:” por “,”, y las minúsculas por mayúsculas [5].

- Master:

```
AT + ROLE = 1     //ROLE = 0 => Slave;   ROLE = 1 => Master
AT + CMODE = 0    //Entrar a modo de conexión: este comando es el que da
                  //pie o permite ejecutar el siguiente comando, que
                  //tiene que ir a continuación de este

AT + BIND = <Dirección MAC copiada del Slave> //Configurar dirección
//MAC del slave que este master tiene que buscar cuando se encienda
```

Tras ejecutar con éxito estos comandos, podemos desconectar la alimentación de Master y Slave, que habrán quedado correctamente configurados.

6. MANUAL DE USUARIO DEL ROBOT RECOGEDOR

Para inicializar este robot recogedor, simplemente hemos de seguir los siguientes pasos:

1. Conectar los terminales de la batería del robot y la toma de alimentación general del mismo. De esta manera, todos los circuitos quedan alimentados, y deberán encenderse los leds de todos los drivers y del microcontrolador. Por su parte, el módulo Bluetooth del robot quedará parpadeando rápidamente con un led rojo, indicando que este módulo está buscando al módulo Bluetooth del mando para comenzar la conexión mando-robot.

Figura 30. Pantalla de bienvenida en el mando de control

2. Conectar el mando Bluetooth a un PC por USB. Aparecerá la siguiente pantalla de bienvenida. Para comenzar con la funcionalidad, simplemente tocamos una vez la pantalla. Por otro lado, al conectar el mando, el led rojo y el azul de su módulo Bluetooth comenzarán a parpadear rápidamente, indicando que está buscando al módulo Bluetooth del robot. Debemos esperar a que esta conexión se establezca. Sabremos que esto ha ocurrido cuando ambos módulos comiencen a parpadear lentamente.

3. En este momento, el robot ya está recibiendo las órdenes del mando. La pantalla del mando debería ser la siguiente, en la que encontraremos los siguientes elementos:
    1. Dibujo interactivo del robot (elegir articulación a manejar)
    2. Dibujo del estado de los joystick + ayudas en pantalla

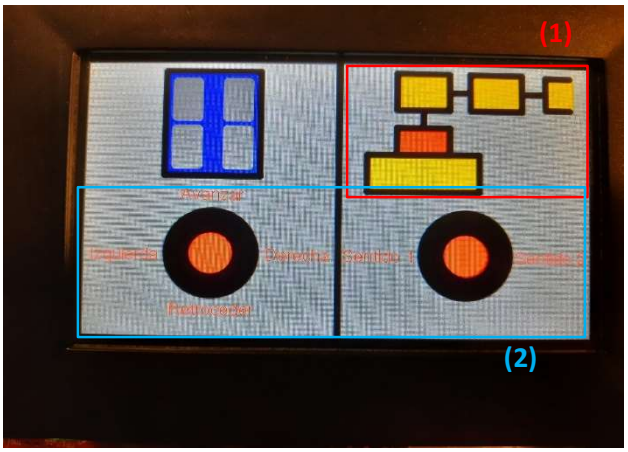


Figura 31. Interfaz de usuario en el mando de control del robot

Como podemos ver en la pantalla, el control del brazo y del coche se realizan de forma separada, y por tanto, podríamos operar ambos simultáneamente (el robot permite mover su manipulador a la vez que la plataforma se desplaza). Los controles se reparten de la siguiente manera:

- **Control de la plataforma:** se realiza con el joystick izquierdo del mando. El eje Y nos sirve para avanzar y retroceder, mientras que el eje X sirve para girar a derecha e izquierda, tal y como nos indica la ayuda en la pantalla.
  El avance de la plataforma está condicionado por un sensor de distancia ultrasonidos que tiene en su parte delantera. Por su parte, el movimiento del manipulador no está condicionado por dicho sensor, y tampoco lo están los giros o el retroceso de la plataforma, para así poder salvar el posible obstáculo frontal.

- **Control del manipulador:** se realiza con el joystick derecho del mando y de forma táctil a través del dibujo del robot en la pantalla. En la figura, se nos mostrará en rojo la articulación del manipulador que se está controlando con el joystick (torreta giratoria, hombro, codo, muñeca o garra), dado que el joystick sólo puede controlar una articulación a la vez.
  El movimiento de la articulación se realiza con el eje X del joystick: con la mitad izquierda del eje X del joystick haremos que la articulación se mueva en un sentido, y con la mitad derecha haremos que la articulación se mueva en el otro.
  Con estos movimientos, podremos ajustar el manipulador y la garra manualmente, para poder manipular el objeto con la garra. Recordamos que el manipulador soporta un peso máximo de 100 g.

Finalmente, para apagar el robot, simplemente desconectamos su conector de alimentación general. Además, es recomendable desconectar al menos uno de los terminales de la batería. Por su parte, para apagar el mando simplemente lo desconectamos del puerto USB del ordenador.

7. ANEXOS DE PROGRAMAS

7.1. CÓDIGO "RobotFinal.c"

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "driverlib2.h"

#include "LibreriaRobotSEPA.h"


uint32_t reloj=0;                       //Reloj del sistema
int PeriodoPWM;                         //Periodo para que los PWM se configuren a 50 Hz
volatile int Max_vel = 40000;           //Valor máximo del PWM determinado experimentalmente para que la velocidad del coche sea máxima
volatile int Min_vel = 1000;            //Valor mínimo del PWM de tal manera que la velocidad del coche es mínima pero no nula (determinada experimentalmente)
int velCoche;                           //Variable que almacena el periodo PWM para configurar la velocidad del coche
int velBrazo;                           //Variable que almacena el periodo PWM para configurar la velocidad del brazo
int PuertoIN1,PinIN1,PuertoIN2,PinIN2,OutPWM;       //Variables para almacenar

int MotorElegido = 2;                   //Valor que almacena la articulación del brazo que se va a mover con el joystick
//Su valor vendrá dado por el valor que le indique el mando

int mMenos,mMas;                        //Pendientes de las rectas de mapeos de velocidades de coche y brazo
int bMenos,bMas;                        //Ordenadas en el origen de las rectas de mapeos de velocidades de coche y brazo
int EjeXIzda=0, EjeYIzda=0, EjeXDcha=0, EjeYDcha=0;     //Variables donde se almacenan las lecturas de los joystick del mando

volatile int UARTflag=0;                //Flag de interrupción que se activa cuando se recibe una transferencia de datos
//completa del mando (cada ciclo incluye las lecturas de los 2 ejes de cada joystick,
//y la articulación del brazo elegida para moverse con el joystick del brazo)

volatile int i=0;                       //Variable para recorrer el buffer de recepción de la comunicación serie. Debe estar fuera
//de la rutina de interrupción para mantener el valor entre distintas recepciones (interrupciones)

volatile int buffer[10]={0,0,0,0,0,0,0,0,0,0};          //Buffer para almacenar una transmisión completa del mando. El mando envía en este orden:
```

```
//Caracter de control, EjeXIzda8bitsMásSig,EjeXIzda8bitsMenosSig,
EjeYIzda8bitsMásSig,EjeYIzda8bitsMenosSig,
//EjeXDcha8bitsMásSig,EjeXDcha8bitsMenosSig,
EjeYDcha8bitsMásSig,EjeYDcha8bitsMenosSig, MotorElegido.
//El caracter de control ('A') sirve para sincronizar la emisión y la recepción.

volatile int d=0;                    //Distancia en cm medida por el ultrasonidos
volatile int t=0;                    //Tiempo en microsegundos que cuenta el Timer1
volatile int t1=0;                   //Tiempo en microsegundos que tarda el echo en pasar de
0 a 1 (medir el ancho de echo)
volatile int t2=0;                   //Tiempo en microsegundos que tarda el echo en pasar de
1 a 0 (medir el ancho de echo)
volatile int flag=0;                 //Flag de interrupción para la rutina de interrupción del
ultrasonidos


//RUTINA DE INTERRUPCIÓN TIMER1 (ULTRASONIDOS): contar tiempo en microsegundos
void IntUSeg(void)
{
TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT); // Borra flag
t++;
}

//RUTINA DE INTERRUPCIÓN ECHO (ULTRASONIDOS): su finalidad es medir el ancho de la onda
de echo que se recibe (medir incremento t2-t1)
void rutina_interrupcionEcho(void)
{
if(flag==0 && GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_7))
{
flag=1;
t1=t;
}
else if (flag==1 && !GPIOPinRead(GPIO_PORTC_BASE,GPIO_PIN_7))
{
flag=2;
t2=t;
}

GPIOIntClear(GPIO_PORTC_BASE, GPIO_PIN_7);
}

//FUNCIÓN DE INTERRUPCIÓN DE LA UART6: interrumpirá al micro cada vez que llegue un dato
discreto
//(no una transmisión completa) del mando (Bluetooth)
void IntUART(void)
{
buffer[i]=UARTCharGet(UART6_BASE);
if(buffer[0]==65)
{
i++;
}
if(i>9)
```

```
{
UARTflag=1;
i=0;
}
UARTIntClear(UART6_BASE, UART_INT_RX); // Borra flag

}


int main(void){

reloj=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
SYSCTL_CFG_VCO_480), 120000000);



//HABILITAR PUERTOS GPIO
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);

//CONFIGURAR UART0 PARA COMUNICACIÓN SERIE CON EL ORDENADOR (DEPURACIÓN)
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
UARTStdioConfig(0, 115200, reloj);

//CONFIGURAR UART6 PARA COMUNICACIÓN BLUETOOTH (SLAVE)
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART6);
GPIOPinConfigure(GPIO_PP0_U6RX);
GPIOPinConfigure(GPIO_PP1_U6TX);
GPIOPinTypeUART(GPIO_PORTP_BASE, GPIO_PIN_0 | GPIO_PIN_1);

//UARTStdioConfig(6, 115200, reloj);  //La UART que use el bluetooth se tiene que configurar a
mano para que no use la librería uartstdio.c
UARTConfigSetExpClk(UART6_BASE, reloj, 115200,
(UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
UART_CONFIG_WLEN_8));
UARTEnable(UART6_BASE);

//CONFIGURACIÓN DE TODOS LOS PWM
```

```
//Configuracion PWM a través de una función de la librería "LibreriaRobotSEPA.c", que incluye las líneas necesarias para
//configurar un PWM a 50 Hz
configuraPWM50Hz(SYSCTL_PERIPH_PWM0, SYSCTL_PERIPH_GPIOF, PWM0_BASE, GPIO_PF1_M0PWM1, GPIO_PORTF_BASE, GPIO_PIN_1, PWM_GEN_0, PWM_OUT_1, PWM_OUT_1_BIT);
configuraPWM50Hz(SYSCTL_PERIPH_PWM0, SYSCTL_PERIPH_GPIOF, PWM0_BASE, GPIO_PF2_M0PWM2, GPIO_PORTF_BASE, GPIO_PIN_2, PWM_GEN_1, PWM_OUT_2, PWM_OUT_2_BIT);
configuraPWM50Hz(SYSCTL_PERIPH_PWM0, SYSCTL_PERIPH_GPIOF, PWM0_BASE, GPIO_PF3_M0PWM3, GPIO_PORTF_BASE, GPIO_PIN_3, PWM_GEN_1, PWM_OUT_3, PWM_OUT_3_BIT);
configuraPWM50Hz(SYSCTL_PERIPH_PWM0, SYSCTL_PERIPH_GPIOG, PWM0_BASE, GPIO_PG0_M0PWM4, GPIO_PORTG_BASE, GPIO_PIN_0, PWM_GEN_2, PWM_OUT_4, PWM_OUT_4_BIT);
configuraPWM50Hz(SYSCTL_PERIPH_PWM0, SYSCTL_PERIPH_GPIOG, PWM0_BASE, GPIO_PG1_M0PWM5, GPIO_PORTG_BASE, GPIO_PIN_1, PWM_GEN_2, PWM_OUT_5, PWM_OUT_5_BIT);
configuraPWM50Hz(SYSCTL_PERIPH_PWM0, SYSCTL_PERIPH_GPIOK, PWM0_BASE, GPIO_PK4_M0PWM6, GPIO_PORTK_BASE, GPIO_PIN_4, PWM_GEN_3, PWM_OUT_6, PWM_OUT_6_BIT);
configuraPWM50Hz(SYSCTL_PERIPH_PWM0, SYSCTL_PERIPH_GPIOK, PWM0_BASE, GPIO_PK5_M0PWM7, GPIO_PORTK_BASE, GPIO_PIN_5, PWM_GEN_3, PWM_OUT_7, PWM_OUT_7_BIT);

//CONFIGURACIÓN DE LOS MAPEOS
//Mapeo: usaremos estas pendientes y ordenadas en el origen para mapear las lecturas de los joystick a velocidades de los motores
mMenos=(Min_vel-Max_vel)/1850;
mMas=(Max_vel-Min_vel)/(4096-2300);
bMenos=Max_vel;
bMas=Min_vel-mMas*2300;
////////////////////////////


//CONFIGURACIÓN DE TODAS LAS SEÑALES DIGITALES
//SALIDAS MOTORES DEL BRAZO
GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_0|GPIO_PIN_1);      //MOTOR 1 IN1_M1 = PE0   IN2_M1 = PE1
GPIOPinTypeGPIOOutput(GPIO_PORTL_BASE, GPIO_PIN_4|GPIO_PIN_5); //MOTOR 2 IN1_M2 = PL4   IN2_M2 = PL5
GPIOPinTypeGPIOOutput(GPIO_PORTM_BASE, GPIO_PIN_7);         //MOTOR 3 IN1_M3 = PM7
GPIOPinTypeGPIOOutput(GPIO_PORTP_BASE, GPIO_PIN_5);         //MOTOR 3 IN2_M3 = PP5
GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_7);         //MOTOR 4 IN1_M4 = PA7
GPIOPinTypeGPIOOutput(GPIO_PORTM_BASE, GPIO_PIN_0);         //MOTOR 4 IN2_M4 = PM0
GPIOPinTypeGPIOOutput(GPIO_PORTK_BASE, GPIO_PIN_2|GPIO_PIN_3);      //MOTOR 5 IN1_M5 = PK2   IN2_M5 = PK3

//SALIDAS MOTORES DEL COCHE
```

```
GPIOPinTypeGPIOOutput(GPIO_PORTH_BASE, GPIO_PIN_2|GPIO_PIN_3);    //MOTOR IZDA IN1_IZDA = PH3   IN2_IDZA = PH2
GPIOPinTypeGPIOOutput(GPIO_PORTE_BASE, GPIO_PIN_3 | GPIO_PIN_5);//MOTOR DCHA IN1_DCHA = PD7   IN2_DCHA = PE3

//ENTRADAS DEL BLUETOOTH: ya configuradas en la UART6

//ENTRADA Y SALIDA DEL ULTRASONIDOS
GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_7); //Pin echo
GPIOPadConfigSet(GPIO_PORTC_BASE,GPIO_PIN_7,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU);
GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_6); //Pin trigger

//CONFIGURACIÓN DEL TIMER1 (ULTRASONIDOS)
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1); //Habilita T1
TimerClockSourceSet(TIMER1_BASE, TIMER_CLOCK_SYSTEM); //T1 a 120MHz
TimerConfigure(TIMER1_BASE, TIMER_CFG_PERIODIC); //T1 periodico y 32bits
TimerLoadSet(TIMER1_BASE, TIMER_A, reloj/1000000-1);

//CONFIGURAR INTERRUPCIÓN DE LA UART
UARTIntRegister(UART6_BASE,IntUART);
IntEnable(INT_UART6);
UARTIntEnable(UART6_BASE, UART_INT_RX);
IntMasterEnable();

//CONFIGURAR INTERRUPCIÓN PIN ECHO ULTRASONIDOS
GPIOIntTypeSet(GPIO_PORTC_BASE,GPIO_PIN_7,GPIO_BOTH_EDGES);  // Definir tipo int: flancos de subida y de bajada
GPIOIntEnable(GPIO_PORTC_BASE, GPIO_PIN_7);          // Habilitar pin de interrupción: pin echo, PC7
GPIOIntRegister(GPIO_PORTC_BASE, rutina_interrupcionEcho);  //Registrar (definir) la rutina de interrupción
IntEnable(INT_GPIOC);                    //Habilitar interrupción del pto C

//CONFIGURAR INTERRUPCIÓN DEL TIMER1
TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
TimerIntRegister(TIMER1_BASE,TIMER_A,IntUSeg);
IntEnable(INT_TIMER1A);
IntMasterEnable();

//CONFIGURAR PERIFÉRICOS QUE QUEDARÁN ACTIVOS DURANTE EL MODO DE BAJO CONSUMO
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOC);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOD);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOE);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOF);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOG);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOH);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOP);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_PWM0);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_UART6);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_UART0);
```

```
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_TIMER1);
SysCtlPeripheralClockGating(true);

//BUCLE INFINITO
d=0;              //Inicializar distancia del ultrasonidos a 0 en cada iteración
IntDisable(INT_GPIOC);  //Deshabilitar interrupciones del echo del ultrasonidos mientras no se esté usando

while(1)
{
while(UARTflag==0)SysCtlSleep();  //Mientras la UART6 no reciba nada, estamos en modo sleep
UARTflag=0;                 //Justo cuando el micro se despierte, desactivamos el flag de la UART

EjeXIzda=((buffer[1]<<8) | (buffer[2]));          //Si en la rutina de interrupción se han recibido todas las órdenes
EjeYIzda=((buffer[3]<<8) | (buffer[4]));         //del mando en una cadena llamada "buffer", aquí la deshacemos para
EjeXDcha=((buffer[5]<<8) | (buffer[6]));          //obtener todas las variables que ha enviado el mando
EjeYDcha=((buffer[7]<<8) | (buffer[8]));
MotorElegido=buffer[9];

//MONITORIZACIÓN DE VARIABLES POR LA UART0 (COMUNICACIÓN SERIE CON EL ORDENADOR)
//Borrar pantalla
UARTprintf("\033[8;1H                                                    \n");
UARTprintf("                                                          ");
//Sacar por UART0 las posiciones de los mandos recibidas
UARTprintf("\033[8;1HEJEXIZDA = %d EJEYIZDA = %d MOTORELEGIDO = %d              \n",
EjeXIzda,EjeYIzda,MotorElegido);
UARTprintf("EJEXDCHA = %d EJEYDCHA = %d  VelCoche=%d                          ",
EjeXDcha,EjeYDcha,velCoche);

//CONTROL DEL ROBOT: AVANCE DEL COCHE CONDICIONADO A QUE EL ULTRASONIDOS DETECTE UNA DISTANCIA MAYOR QUE 15 cm
//DE ESTA FORMA SE EVITAN COLISIONES DEL ROBOT

//El giro y el retroceso siguen permitidos, para salvar el obstáculo

//CONTROL COCHE: la dirección del coche se decidirá con el joystick de la izquierda
if(EjeYIzda < 1850){                //Si el joystick está en la parte superior, avanzar
if(d>15) velCoche=mMenos*EjeYIzda+bMenos;  //Mapear velocidad del coche con el eje del Joystick
else velCoche = 10;                 //Dar al coche velocidad nula para avanzar si tengo un objeto delante
Avanzar(velCoche);                  //Función de la librería "LibreriaRobotSEPA.c"
}

if(EjeYIzda > 2300){          //Si el joystick está en la parte inferior, retroceder
velCoche=mMas*EjeYIzda+bMas;  //Mapear velocidad del coche con el eje del Joystick
Retroceder(velCoche);       //Función de la librería "LibreriaRobotSEPA.c"
}
```

```
if(EjeYIzda > 1850 && EjeYIzda < 2300 && EjeXIzda > 1850 && EjeXIzda < 2300){  //Si el joystick está en la posición central, parar el coche
velCoche=0;                                          //Realmente Parar() hace que velCoche = 10 (cantidad determinada
//experimentalmente para detenerse), pero dejamos aquí esta variable
//monitorizar por puerto serie.
Parar();                                             //Función de la librería "LibreriaRobotSEPA.c"
}

if(EjeXIzda > 2300){          //Si el joystick está en la parte derecha, girar a la derecha
velCoche=mMas*EjeXIzda+bMas;  //Mapear velocidad del coche con el eje del Joystick
GiroDerecha(velCoche);        //Función de la librería "LibreriaRobotSEPA.c"
}

if(EjeXIzda < 1850){          //Si el joystick está en la parte izquierda, girar a la izquierda
velCoche=mMenos*EjeXIzda+bMenos;  //Mapear velocidad del coche con el eje del Joystick
GiroIzquierda(velCoche);      //Función de la librería "LibreriaRobotSEPA.c"
}


//CONTROL BRAZO

//En función del motor elegido, en el control que viene a continuación actuaremos sobre una articulación u otra.
//Para ello, inicializaremos estas variables intermedias, que usaremos en el código de control que aparece a continuación.
switch(MotorElegido){
case 1:
PuertoIN1 = GPIO_PORTE_BASE; //E0
PuertoIN2 = GPIO_PORTE_BASE; //E1
PinIN1 = GPIO_PIN_0;
PinIN2 = GPIO_PIN_1;
OutPWM = PWM_OUT_2;
break;

case 2:
PuertoIN1 = GPIO_PORTL_BASE;   //L4
PuertoIN2 = GPIO_PORTL_BASE;   //L5
PinIN1 = GPIO_PIN_4;
PinIN2 = GPIO_PIN_5;
OutPWM = PWM_OUT_3;
break;

case 3:
PuertoIN1 = GPIO_PORTM_BASE;  //M7
PuertoIN2 = GPIO_PORTP_BASE;  //P5
PinIN1 = GPIO_PIN_7;
PinIN2 = GPIO_PIN_5;
OutPWM = PWM_OUT_5;
break;
```

```
case 4:
PuertoIN1 = GPIO_PORTA_BASE;  //A7
PuertoIN2 = GPIO_PORTM_BASE;  //M0
PinIN1 = GPIO_PIN_7;
PinIN2 = GPIO_PIN_0;
OutPWM = PWM_OUT_6;
break;

case 5:
PuertoIN1 = GPIO_PORTK_BASE;  //K2
PuertoIN2 = GPIO_PORTK_BASE;  //K3
PinIN1 = GPIO_PIN_2;
PinIN2 = GPIO_PIN_3;
OutPWM = PWM_OUT_7;
break;
}

//Código de control de la articulación x (x = de 1 a 5) del brazo del robot:
//La velocidad del motor x se decidirá con el eje horizontal del joystick, y el motor a mover a través de la pantalla del mando
//Control motor x (eje horizontal)
if(EjeXDcha > 2300){  //Si la velocidad es positiva, girar en un sentido
GPIOPinWrite(PuertoIN1,PinIN1,PinIN1);       //IN1Dcha a 1
GPIOPinWrite(PuertoIN2,PinIN2,0);           //IN2Dcha a 0
velBrazo=mMas*EjeXDcha+bMas;                //Mapear velocidad del coche con el eje del Joystick
PWMPulseWidthSet(PWM0_BASE, OutPWM, velBrazo); //Dar velocidad al motor x del brazo
}

if(EjeXDcha < 1850){                        //Si la velocidad es negativa, girar en el otro sentido
GPIOPinWrite(PuertoIN1,PinIN1,0);           //IN1Dcha a 0
GPIOPinWrite(PuertoIN2,PinIN2,PinIN2);       //IN2Dcha a 1
velBrazo=mMenos*EjeXDcha+bMenos;                //Mapear velocidad del coche con el eje del Joystick
PWMPulseWidthSet(PWM0_BASE, OutPWM, velBrazo); //Dar velocidad al motor x del brazo
}

if(EjeXDcha > 1850 && EjeXDcha < 2300){         //Si el joystick está en la posición central, parar el motor
//GPIOPinWrite(PuertoIN1,PinIN1,0);            //IN1Dcha a 0   Da igual el sentido del motor, su velocidad será cero
//GPIOPinWrite(PuertoIN2,PinIN2,PinIN2);      //IN2Dcha a 1
velBrazo = 10;                          //Valor experimental del PWM para detener el motor
PWMPulseWidthSet(PWM0_BASE, OutPWM, velBrazo); //Dar velocidad al motor x del brazo
}

//MANEJO DEL SENSOR ULTRASONIDOS
flag=0;                                 //Bajar el flag implica inicio de la medida de distancia
GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_6,0);          //Poner a 0 Trigger
TimerEnable(TIMER1_BASE, TIMER_A);                //Comenzar a contar el tiempo
while(t<4)SysCtlSleep();                    //Pondremos el Trigger durante 4us para asegurar un disparo limpio
TimerDisable(TIMER1_BASE, TIMER_A);               //Detener la cuenta de tiempo
```

```
t=0;                                    //Tras un comienzo de disparo limpio, reseteamos el tiempo y
lo medimos
//para generación del pulso del Trigger (por ejemplo, 10 us)

GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_6,GPIO_PIN_6);   //Poner a 1 Trigger
TimerEnable(TIMER1_BASE, TIMER_A);                  //Comenzar a contar el tiempo
while(t<10)SysCtlSleep();                       //Durante 10 us, mantendremos Trigger a 1
TimerDisable(TIMER1_BASE, TIMER_A);                 //Detener la cuenta de tiempo
GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_6,0);             //Poner a 0 Trigger

t=0;                                    //Reseteamos la variable tiempo para ahora usarla para el
cálculo de la distancia

TimerEnable(TIMER1_BASE, TIMER_A);                  //Comenzar a contar el tiempo
IntEnable(INT_GPIOC);                           //Habilitar interrupción del echo del ultrasonidos
while(t<11800 && (flag==0 || flag==1))SysCtlSleep();    //11800 useg es el máximo tiepo que
esperamos echo, si no se recibe en ese tiempo,
//entonces distancia mayor a 2m (rango sensor) vsonido=343m/seg

IntDisable(INT_GPIOC);                          //Deshabilitar interrupciones del echo del ultrasonidos
mientras no se esté usando
TimerDisable(TIMER1_BASE, TIMER_A);                 //Detener la cuenta del tiempo

//Si el flag de la interrupción del ultrasonidos vale finalmente dos tras la rutina de interrupción,
quiere decir
//que el echo ha vuelto a tiempo al módulo de ultrasonidos. Si tiene otro valor, es que no ha
regresado, y hemos salido
//del SysCtlSleep anterior por timeout.
if(flag==2)
{
d=(t2-t1)/59;  //Cálculo de la distancia (ver memoria)
UARTprintf("Distancia: %d                                   \n",d);
}
else
{
d=200;      //Distancia de dos metros o superior
UARTprintf("Distancia: Distancia superior a 2m                          \n");
}
t=0;            //Resetear el tiempo para la siguiente iteración
}
}
```

7.2. CÓDIGO “MandoBTFinal.c”

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
```

```
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "driverlib2.h"
#include "FT800_TIVA.h"



//Variables para mapear el valor del joystick para dibujar una copia del joystick en pantalla
float mIzdaX, mIzdaY, mDchaX, mDchaY, bIzdaX, bIzdaY, bDchaX, bDchaY;    //Pendientes y ordenadas en el origen para mapear
float MaxEje, MinEje;                                   //Valores máximo y mínimo de los ejes horizontal y vertical de los joystick
int PixelX, PixelY;                                     //Posición final donde se dibuja el joystick en pantalla

//Variables que se envían por comunicación serie (Bluetooth)
int DatoXDcha;
int DatoYDcha;
int DatoXIzda;
int DatoYIzda;
unsigned char MotorElegido = 1;
////////////////////////////////////////////
uint32_t RELOJ=0; //Reloj del sistema

unsigned char ADC0flag = 0; //Flag para rutina de interrupción del ADC0


// =================================================================
// INICIALIZACIÓN DE VARIABLES PARA EL MANEJO DE LA PANTALLA
// =================================================================
#define dword long
#define byte char



#define PosMin 750
#define PosMax 1000

#define XpMax 286
#define XpMin 224
#define YpMax 186
#define YpMin 54

unsigned int Yp=120, Xp=245;
// =================================================================
// Variable Declarations
// =================================================================

char chipid = 0;                // Holds value of Chip ID read from the FT800

unsigned long cmdBufferRd = 0x00000000;      // Store the value read from the REG_CMD_READ register
```

```
unsigned long cmdBufferWr = 0x00000000;             // Store the value read from the
REG_CMD_WRITE register
unsigned int t=0;
//
################################################################################
################################
// User Application - Initialization of MCU / FT800 / Display
//
################################################################################
################################

unsigned long POSX, POSY, BufferXY;
unsigned long POSYANT=0;
unsigned int CMD_Offset = 0;
unsigned long REG_TT[6];
const int32_t REG_CAL[6]={21696,-78,-614558,498,-17021,15755638};
const int32_t REG_CAL5[6]={32146, -1428, -331110, -40, -18930, 18321010};
#define NUM_SSI_DATA          3
////////////////////////////////////////////////////////////////////////////////////////////////
//////////

//INTERRUPCIÓN DEL ADC0: interrumpe cada vez que se termina una conversión de todos los
canales
void InterrADC0(void){
while(!ADCIntStatus(ADC0_BASE, 1, false))       //Esperar a terminar la conversión de todos los
canales
{
}

ADC0flag = 1;
ADCIntClear(ADC0_BASE, 1);    // Poner a 0 el flag de interrupción interno del ADC0

}

int main(void){
uint32_t pui32ADC0Value[4];    //Cadena donde se almacenan todas las conversiones del ADC.
Se declara de tamaño 4
//porque leeremos 4 canales, a los que conectaremos los ejes X e Y de los dos joystick
//(un eje a cada canal)

RELOJ=SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ | SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
SYSCTL_CFG_VCO_480), 120000000);
HAL_Init_SPI(2, RELOJ);  //Boosterpack a usar, Velocidad del MC
Inicia_pantalla();      //Arranque de la pantalla

//CONFIGURAR UART0 PARA COMUNICACIÓN SERIE CON EL ORDENADOR (DEPURACIÓN)
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
GPIOPinConfigure(GPIO_PA0_U0RX);
GPIOPinConfigure(GPIO_PA1_U0TX);
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
```

```
UARTStdioConfig(0, 115200, RELOJ);  //Estamos configurando la UART0 con funciones de uartstdio para poder usar printf en PuTTY

//CONFIGURAR UART5 PARA COMUNICACIÓN BLUETOOTH (MASTER)
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART5);
GPIOPinConfigure(GPIO_PC6_U5RX);
GPIOPinConfigure(GPIO_PC7_U5TX);
GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_6 | GPIO_PIN_7);

//UARTStdioConfig(5, 115200, reloj);  //La UART que use el bluetooth se tiene que configurar a mano para que no use la librería uartstdio.c
UARTConfigSetExpClk(UART5_BASE, RELOJ, 115200,
(UART_CONFIG_PAR_NONE | UART_CONFIG_STOP_ONE |
UART_CONFIG_WLEN_8));
UARTEnable(UART5_BASE);

//CONFIGURACIÓN ADC0:
//JOYSTICK SITUADO A LA IZDA:   VRx = PE3    VRy = PE2
//JOYSTICK SITUADO A LA DCHA:   VRx = PE1    VRy = PE0
SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2 | GPIO_PIN_3);          //E2 => eje y dcha; E3 => eje x dcha
GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_0 | GPIO_PIN_1);          //E0 => eje y IZDA; E1 => eje x IZDA (NUEVO)
//A partir de aquí, aparece en todas las funciones del ADC el número 1. Este número es el Sequence Number, y representa el
//número de canales que se van a convertir. Según la documentación, un Sequence Number = 1 nos permite convertir hasta 4 canales.
ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH0);          //El canal 0 (pin AIN0) será para el eje x IZDA
ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH1);          //El canal 1 (pin AIN1) será para el eje y IZDA
ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH2);          //El canal 2 (pin AIN2) será para el eje x DCHA
ADCSequenceStepConfigure(ADC0_BASE, 1, 3, ADC_CTL_CH3 | ADC_CTL_IE |ADC_CTL_END);
//El canal 3 (pin AIN3) será para el eje y DCHA, y cerrará la conversión
ADCSequenceEnable(ADC0_BASE, 1);
ADCIntClear(ADC0_BASE, 1);                                       // Poner a 0 el flag de interrupción interno del ADC0

//CONFIGURACIÓN DE LA RUTINA DE INTERRUPCIÓN DEL ADC0
ADCIntRegister(ADC0_BASE,1,InterrADC0);
ADCIntEnable(ADC0_BASE, 1); //Habilitar la interrupción para salir de modo sleep
IntMasterEnable();

//CONFIGURACIÓN DEL MODO SLEEP: PERIFÉRICOS QUE SE QUEDARÁN DESPIERTOS
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_ADC0);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOE);
```

```
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOJ);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_GPIOC);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_UART0);
SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_UART5);
SysCtlPeripheralClockGating(true);

//DIBUJAR PANTALLA DE BIENVENIDA
Nueva_pantalla(16,16,16);
ComColor(21,160,6);
ComLineWidth(5);
ComRect(10, 10, HSIZE-10, VSIZE-10, true);
ComColor(65,202,42);
ComRect(12, 12, HSIZE-12, VSIZE-12, true);
ComColor(255,255,255);

ComTXT(HSIZE/2,VSIZE/5, 28, OPT_CENTERX,"ROBOT RECOGEDOR");
ComTXT(HSIZE/2,50+VSIZE/5, 28, OPT_CENTERX," SEPA 4 GIERM. 2022 ");
ComTXT(HSIZE/2,100+VSIZE/5, 27, OPT_CENTERX,"Jose Antonio Heredia, Arturo Ubeda");

ComRect(40,40, HSIZE-40, VSIZE-40, false);

Dibuja();
Espera_pant();
int i;
#ifdef VM800B35
for(i=0;i<6;i++)    Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL[i]);
#endif
#ifdef VM800B50
for(i=0;i<6;i++)    Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL5[i]);
#endif

//Mapeo: usaremos estas pendientes y ordenadas en el origen para mapear las lecturas de los joystick a posición en la pantalla
//MaxEje = ValorCentralEje + R - r
//MinEje = ValorCentralEje - R + r
MaxEje = 0.25*HSIZE + 45 - 20;
MinEje = 0.25*HSIZE - 45 + 20;
mIzdaX=(MaxEje-MinEje)/4096;
bIzdaX=MinEje;

MaxEje = 0.7*VSIZE + 45 - 20;
MinEje = 0.7*VSIZE - 45 + 20;
mIzdaY=(MaxEje-MinEje)/4096;
bIzdaY=MinEje;

MaxEje = 0.75*HSIZE + 45 - 20;
MinEje = 0.75*HSIZE - 45 + 20;
mDchaX=(MaxEje-MinEje)/4096;
bDchaX=MinEje;

MaxEje = 0.7*VSIZE + 45 - 20;
MinEje = 0.7*VSIZE - 45 + 20;
```

```
mDchaY=(MaxEje-MinEje)/4096;
bDchaY=MinEje;
////////////////////////////
//BUCLE INFINITO
while(1)
{
ADCProcessorTrigger(ADC0_BASE, 1);              //Comenzar la conversión
while(ADC0flag==0)SysCtlSleep();                //Poner a dormir al micro hasta que no acabe la conversión de todos los canales
ADC0flag = 0;
ADCSequenceDataGet(ADC0_BASE, 1, pui32ADC0Value);    // Almacenar las conversiones del ADC0 en la cadena de conversiones

//Enviar caracter de control de comienzo de transferencia
UARTCharPut(UART5_BASE,'A');

//JOYSTICK IZDA
//Enviar EjeX izda
UARTprintf("EjeXIzda/16 : %d ",DatoXIzda);
DatoXIzda = pui32ADC0Value[0];
UARTCharPut(UART5_BASE,(DatoXIzda >> 8)& 0xff);     //Envio 8 bits más significativos EjeXIz
UARTCharPut(UART5_BASE,DatoXIzda & 0xff);          //Envio 8 bits menos significativos EjeXIz
//Enviar EjeY IZDA
DatoYIzda = pui32ADC0Value[1];
UARTprintf("\t\tEjeYIzda/16 Enviado: %d", DatoYIzda);
UARTCharPut(UART5_BASE,(DatoYIzda >> 8)& 0xff);     //Envio 8 bits más significativos EjeYIz
UARTCharPut(UART5_BASE,DatoYIzda & 0xff);          //Envio 8 bits menos significativos EjeYIz
//JOYSTICK DCHA
//Enviar EjeX DCHA
DatoXDcha = pui32ADC0Value[2];
UARTprintf("\t\tEjeXDcha/16 Enviado: %d ", DatoXDcha);
UARTCharPut(UART5_BASE, (DatoXDcha >> 8)& 0xff);    //Envio 8 bits más significativos EjeXDer
UARTCharPut(UART5_BASE, DatoXDcha & 0xff);         //Envio 8 bits menos significativos EjeXDer
//Enviar EjeY DCHA
DatoYDcha = pui32ADC0Value[3];
UARTprintf("\t\tEjeYDcha/16 Enviado: %d",DatoYDcha);
UARTCharPut(UART5_BASE, (DatoYDcha >> 8)& 0xff);    //Envio 8 bits más significativos EjeYDer
UARTCharPut(UART5_BASE, DatoYDcha & 0xff);         //Envio 8 bits menos significativos EjeYDer
//Enviar Motor Elegido
UARTprintf("\t\tMotorElegido: %d\n", MotorElegido);
UARTCharPut(UART5_BASE,MotorElegido);              //Envio Motor Elegido


//MANEJO DE LA PANTALLA
Lee_pantalla();                        //Obtener pulsación de la pantalla
Nueva_pantalla(16,16,16);                    //Tras la pantalla de bienvenida, dibujar la pantalla de manejo del robot

ComGradient(0,0,GRIS_CLARO,0,240,GRIS_OSCURO);
ComColor(255,0,0);
ComFgcolor(200, 200, 10);
```

```
//DIBUJAR LÍNEA DIVISORIA
ComColor(0,0,0); //Negro
ComLine(0.5*HSIZE, 0, 0.5*HSIZE, VSIZE, 3);

//DIBUJO DE LAS POSICIONES DE LOS JOYSTICK MAPEADAS EN LA PANTALLA
//JOYSTICK IZQUIERDO
ComColor(0,0,0);
ComCirculo(0.25*HSIZE, 0.7*VSIZE, 45);

ComColor(255,0,0);
PixelX=mIzdaX*DatoXIzda+bIzdaX;
PixelY=mIzdaY*DatoYIzda+bIzdaY;
ComCirculo(PixelX, PixelY, 20);        //Dibujar joystick en la posición mapeada

ComTXT(0.25*HSIZE, 0.45*VSIZE, 22, OPT_CENTERX,"Avanzar");
ComTXT(0.25*HSIZE, 0.87*VSIZE, 22, OPT_CENTERX,"Retroceder");
ComTXT(0.08*HSIZE, 0.67*VSIZE, 22, OPT_CENTERX,"Izquierda");
ComTXT(0.42*HSIZE, 0.67*VSIZE, 22, OPT_CENTERX,"Derecha");

//JOYSTICK DERECHO
ComColor(0,0,0);
ComCirculo(0.75*HSIZE, 0.7*VSIZE, 45);

ComColor(255,0,0);
PixelX=mDchaX*DatoXDcha+bDchaX;
PixelY=mDchaY*DatoYDcha+bDchaY;
ComCirculo(PixelX, PixelY, 20);        //Dibujar joystick en la posición mapeada

ComTXT(0.58*HSIZE, 0.67*VSIZE, 22, OPT_CENTERX,"Sentido 1");
ComTXT(0.92*HSIZE, 0.67*VSIZE, 22, OPT_CENTERX,"Sentido 2");

//DIBUJAR BRAZO DEL ROBOT
ComColor(0,0,0); //Negro
ComLine(0.75*HSIZE-40, 0.25*VSIZE, 0.75*HSIZE-40, 0.05*VSIZE, 3);

ComColor(255,233,0);
ComRect(0.65*HSIZE-40,0.35*VSIZE,0.85*HSIZE-40,0.45*VSIZE,true);      //Rectángulo torreta
(Motor 5)
ComRect(0.75*HSIZE-20-40,0.25*VSIZE,0.75*HSIZE+20-40,0.35*VSIZE,true);        //Rectángulo
hombro (Motor 4)

ComColor(0,0,0); //Negro
ComLine(0.75*HSIZE-40, 0.1*VSIZE, HSIZE-40, 0.1*VSIZE, 3);

ComColor(255,233,0);  //Amarillo
ComRect(0.75*HSIZE-20-40,0.05*VSIZE,0.75*HSIZE+20-40,0.15*VSIZE,true);        //Rectángulo
codo (Motor 3)
ComRect(0.75*HSIZE-40+50,0.05*VSIZE,0.75*HSIZE-40+90,0.15*VSIZE,true);        //Rectángulo
muñeca (Motor 2)

ComRect(HSIZE-40,0.05*VSIZE,HSIZE-40+20,0.15*VSIZE,true); //Rectángulo garra (Motor 1)
ComColor(0,0,0); //Negro
```

```
ComLine(HSIZE-42, 0.05*VSIZE, HSIZE-42, 0.15*VSIZE, 3);
ComLine(HSIZE-42, 0.05*VSIZE-3, HSIZE-18, 0.05*VSIZE-3, 3);
ComLine(HSIZE-42, 0.15*VSIZE+3, HSIZE-18, 0.15*VSIZE+3, 3);

//ELEGIR QUÉ ARTICULACIÓN SE DIBUJA DE ROJO
ComColor(255,0,0);
switch(MotorElegido){
case 1:
ComRect(HSIZE-40,0.05*VSIZE,HSIZE-40+20,0.15*VSIZE,true); //Rectángulo garra (Motor 1)
break;
case 2:
ComRect(0.75*HSIZE-40+50,0.05*VSIZE,0.75*HSIZE-40+90,0.15*VSIZE,true);        //Rectángulo
muñeca (Motor 2)
break;
case 3:
ComRect(0.75*HSIZE-20-40,0.05*VSIZE,0.75*HSIZE+20-40,0.15*VSIZE,true);        //Rectángulo
codo (Motor 3)
break;
case 4:
ComRect(0.75*HSIZE-20-40,0.25*VSIZE,0.75*HSIZE+20-40,0.35*VSIZE-10,true);    //Rectángulo
hombro (Motor 4)
break;
case 5:
ComRect(0.65*HSIZE-40,0.35*VSIZE,0.85*HSIZE-40,0.45*VSIZE,true);     //Rectángulo  torreta
(Motor 5)
break;
}

//DIBUJO DE BORDES DE CADA RECTÁNGULO Y ACTUALIZACIÓN DE MotorElegido
ComColor(0,0,0);
ComRect(0.65*HSIZE-40-5,0.35*VSIZE-5,0.85*HSIZE-40+5,0.45*VSIZE+5,false);     //Rectángulo
torreta (Motor 5)
if(POSX>0.65*HSIZE-40-5 && POSX<0.85*HSIZE-40+5 && POSY>0.35*VSIZE-5 &&
POSY<0.45*VSIZE+5) //Pulsando en el rectangulo
{
MotorElegido = 5;
}

ComColor(0,0,0);
ComRect(0.75*HSIZE-20-40-5,0.25*VSIZE-5,0.75*HSIZE+20-40+5,0.35*VSIZE-5,false);
if(POSX>0.75*HSIZE-20-40-5 && POSX<0.75*HSIZE+20-40+5 && POSY>0.25*VSIZE-5 &&
POSY<0.35*VSIZE-5) //Pulsando en el rectangulo
{
MotorElegido = 4;
}

ComColor(0,0,0);
ComRect(0.75*HSIZE-20-40-5,0.05*VSIZE-5,0.75*HSIZE+20-40+5,0.15*VSIZE+5,false);
if(POSX>0.75*HSIZE-20-40-5 && POSX<0.75*HSIZE+20-40+5 && POSY>0.05*VSIZE-5 &&
POSY<0.15*VSIZE+5) //Pulsando en el rectangulo
{
MotorElegido = 3;
```

```
}

ComColor(0,0,0);
ComRect(0.75*HSIZE-40+50-5,0.05*VSIZE-5,0.75*HSIZE-40+90+5,0.15*VSIZE+5,false);
if(POSX>0.75*HSIZE-40+50 && POSX<0.75*HSIZE-40+90 && POSY>0.05*VSIZE && POSY<0.15*VSIZE) //Pulsando en el rectangulo
{
MotorElegido = 2;
}

ComColor(0,0,0);
if(POSX>HSIZE-40 && POSX<HSIZE-40+20 && POSY>0.05*VSIZE && POSY<0.15*VSIZE) //Pulsando en el rectangulo
{
MotorElegido = 1;
}

//DIBUJAR COCHE DEL ROBOT
ComColor(0,0,255);
ComRect(0.25*HSIZE-40,0.05*VSIZE,0.25*HSIZE+40,0.40*VSIZE,true);
ComColor(155,155,155);
ComRect(0.20*HSIZE-10,0.1*VSIZE-8,0.20*HSIZE+10,0.1*VSIZE+30-3,true);
ComRect(0.20*HSIZE-10,0.1*VSIZE-10+50,0.20*HSIZE+10,0.1*VSIZE+30-5+50,true);

ComRect(0.30*HSIZE-10,0.1*VSIZE-8,0.30*HSIZE+10,0.1*VSIZE+30-3,true);
ComRect(0.30*HSIZE-10,0.1*VSIZE-10+50,0.30*HSIZE+10,0.1*VSIZE+30-5+50,true);

ComColor(0,0,0);
ComRect(0.25*HSIZE-40-5,0.05*VSIZE-5,0.25*HSIZE+40+5,0.40*VSIZE+5,false);


Dibuja();

}
}
```

7.3. CÓDIGO “LibreriaRobotSEPA.c”

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "driverlib2.h"

void configuraPWM50Hz(uint32_t PeripheralPWM, uint32_t PeripheralGPIO, uint32_t BasePWM, uint32_t PinPWM_GPIO, uint32_t BasePuertoGPIO, uint32_t PinesGPIO, uint32_t GeneradorPWM, uint32_t OutPWM, uint32_t OutPWM_BIT){
```

```
//Configuracion de todos los PWM
SysCtlPeripheralEnable(PeripheralPWM);  //Habilitamos PWM
SysCtlPeripheralEnable(PeripheralGPIO); //Habilitamos puerto pin PWM
PWMClockSet(BasePWM,PWM_SYSCLK_DIV_64);  // al PWM le llega un reloj de 1.875MHz
GPIOPinConfigure(PinPWM_GPIO);        //Configurar el pin a PWM
GPIOPinTypePWM(BasePuertoGPIO, PinesGPIO);
PWMGenConfigure(BasePWM,        GeneradorPWM,        PWM_GEN_MODE_DOWN        |
PWM_GEN_MODE_NO_SYNC);
int PeriodoPWM=37499; // 50Hz  a 1.875MHz

PWMGenPeriodSet(BasePWM, GeneradorPWM, PeriodoPWM); //frec:50Hz
PWMPulseWidthSet(BasePWM, OutPWM, 10);  //Inicialmente, velocidad de motor 0
PWMGenEnable(BasePWM, GeneradorPWM);    //Habilita el generador especificado
PWMOutputState(BasePWM, OutPWM_BIT , true);   //Habilita la salida especificada
}

void Avanzar(int velMotorMapeada){
//Avance del coche: todos los motores (los de la derecha y los de la izquierda) van hacia delante
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5,0);          //IN1Dcha a 0    Debido a que se han
conectado las alimentaciones de los motores
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_3,GPIO_PIN_3);   //IN2Dcha a 1    de los dos lados
del coche de forma contraria, las señales quedan invertidas
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_3,GPIO_PIN_3);    //IN1Izda a 1
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_2,0);          //IN2Izda a 0

PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, velMotorMapeada); //Dar velocidad a los
motores de la DCHA
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, velMotorMapeada); //Dar velocidad a los
motores de la IZDA
}

void Retroceder(int velMotorMapeada){
//Retroceso del coche: todos los motores (los de la derecha y los de la izquierda) van hacia atrás
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5,GPIO_PIN_5);    //IN1Dcha a 1    Debido a que
se han conectado las alimentaciones de los motores
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_3,0);          //IN2Dcha a 0    de los dos lados del
coche de forma contraria, las señales quedan invertidas
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_3,0);          //IN1Izda a 0
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_2,GPIO_PIN_2);    //IN2Izda a 1

PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, velMotorMapeada); //Dar velocidad a los
motores de la DCHA
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, velMotorMapeada); //Dar velocidad a los
motores de la IZDA
}

void GiroDerecha(int velMotorMapeada){
//Giro a la derecha del coche: todos los motores (los de la derecha y los de la izquierda) van
hacia delante,
//pero los motores de la derecha van más rápido que los de la izquierda
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5,GPIO_PIN_5); //IN1Dcha a 1    Debido a que se
han conectado las alimentaciones de los motores
```

```
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_3,0);          //IN2Dcha a 0     de los dos lados del coche de forma contraria, las señales quedan invertidas
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_3,GPIO_PIN_3); //IN1Izda a 1
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_2,0);          //IN2Izda a 0

PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, velMotorMapeada); //Dar velocidad a los motores de la DCHA
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, velMotorMapeada); //Dar velocidad a los motores de la IZDA
}

void GiroIzquierda(int velMotorMapeada){
//Giro a la izquierda del coche: todos los motores (los de la derecha y los de la izquierda) van hacia delante,
//pero los motores de la izquierda van más rápido que los de la derecha
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5,0);          //IN1Dcha a 0     Debido a que se han conectado las alimentaciones de los motores
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_3,GPIO_PIN_3); //IN2Dcha a 1     de los dos lados del coche de forma contraria, las señales quedan invertidas
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_3,0);          //IN1Izda a 0
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_2,GPIO_PIN_2); //IN2Izda a 1

PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, velMotorMapeada); //Dar velocidad a los motores de la DCHA
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, velMotorMapeada); //Dar velocidad a los motores de la IZDA
}

void Parar(){
//Detener el coche: es indiferente los sentidos de giro de los lados del coche: la velocidad es cero para ambos
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_5,0);          //IN1Dcha a 0     Debido a que se han conectado las alimentaciones de los motores
GPIOPinWrite(GPIO_PORTE_BASE,GPIO_PIN_3,GPIO_PIN_3); //IN2Dcha a 1     de los dos lados del coche de forma contraria, las señales quedan invertidas
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_3,0);          //IN1Izda a 0
GPIOPinWrite(GPIO_PORTH_BASE,GPIO_PIN_2,GPIO_PIN_2); //IN2Izda a 1

PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, 10); //Dar velocidad a los motores de la DCHA CON 0 EL COCHE NO SE DETIENE, CON 10 SÍ
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, 10); //Dar velocidad a los motores de la IZDA CON 0 EL COCHE NO SE DETIENE, CON 10 SÍ
}
```

7.4. CÓDIGO "LibreriaRobotSEPA.h"

```
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/adc.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
```

```
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "utils/uartstdio.h"
#include "driverlib2.h"

extern void configuraPWM50Hz(uint32_t PeripheralPWM, uint32_t PeripheralGPIO, uint32_t
BasePWM, uint32_t PinPWM_GPIO, uint32_t BasePuertoGPIO, uint32_t PinesGPIO, uint32_t
GeneradorPWM, uint32_t OutPWM, uint32_t OutPWM_BIT);
extern void Avanzar(int velMotorMapeada);
extern void Retroceder(int velMotorMapeada);
extern void GiroDerecha(int velMotorMapeada);
extern void GiroIzquierda();
extern void Parar();
```

8. REFERENCIAS BIBLIOGRÁFICAS

[1] L. Llamas, «Controlar motores de corriente continua con Arduino y L298N,» Luis Llamas, 26 Mayo 2016. [En línea]. Available: https://www.luisllamas.es/arduino-motor-corriente-continua-l298n/. [Último acceso: 15 Julio 2023].

[2] Electrrio, «EL0513-4 Chasis Robot Arduino 4 Ruedas,» Electrrio, [En línea]. Available: https://www.electrio.es/EL0513-4-Chasis-Robot-Arduino-4-Ruedas. [Último acceso: 24 Junio 2023].

[3] L. Llamas, «Medir distancia con Arduino y sensor de ultrasonidos HC-SR04,» Luis Llamas, 17 Junio 2015. [En línea]. Available: https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/. [Último acceso: 10 Julio 2023].

[4] L. Llamas, «Adaptar 3.3V a 5V (y viceversa) en Arduino con Level Shifter,» Luis Llamas, 4 Enero 2018. [En línea]. Available: https://www.luisllamas.es/arduino-level-shifter/. [Último acceso: 12 Junio 2023].

[5] Naylamp Mechatronics, «Configuración del módulo Bluetooth HC-05 usando comandos AT,» Naylamp Mechatronics, [En línea]. Available: https://naylampmechatronics.com/blog/24_configuracion-del-modulo-bluetooth-hc-05-usando-comandos-at.html. [Último acceso: 19 Junio 2023].