

# ACME Future Engineers 2023

## Cuaderno de ingeniería

### Índice

1.	Introducción .....	2
2.	Fallos del año pasado .....	2
3.	Mejoras .....	2
3.1.	SMD .....	2
3.2.	Encoders .....	3
4.	Programación .....	4
4.1.	Sensores .....	4
4.2.	Microcontroladores 32U4 .....	5
4.3.	SPI .....	5
5.	La placa base .....	6
5.1.	Selección de componentes .....	6
5.2.	Diseño .....	7
5.3.	Creación de la PCB .....	7
5.3.1.	Proceso de insolación y ácidos .....	7
5.3.2.	Máscara de soldadura .....	8
5.3.3.	Proceso de soldado .....	10
6.	La cámara .....	11
7.	Canal de YouTube y videos .....	14
8.	Debugging .....	14
9.	Fotos del vehículo final .....	15

## **1. Introducción**

Somos el equipo ACME, participantes en la WRO Future Engineers 2023.

Este documento tiene la finalidad de mostrar el proceso seguido para crear nuestro vehículo para esta competición, así como incluir imágenes de ciertos momentos en el diseño y la construcción.

## **2. Fallos del año pasado**

Además de participar este año, también participamos el año pasado y el 2021. Como nos basamos en el robot del año pasado tenemos que ver y arreglar los fallos graves que tenía. Uno de los fallos principales era que al usar dos voltajes diferentes (5V y 3,3V) había mucha inestabilidad eléctrica y a veces se llegaba a resetear. Pero estábamos obligados a usar ambos porque el microcontrolador M5Stack usa 3,3V y el Arduino 5V, por esto este año optamos por usar simplemente 5V con microcontroladores 32U4 como si fueran un Arduino normal.

Otro fallo era su velocidad, iba demasiado rápido o mejor dicho, no podía ir lento, esto no nos permitía hacer ciertas maniobras. Para cambiar esto le hemos puesto un engranaje más pequeño al motor para que ahora sea un tercio de la velocidad anterior, de esta manera gana fuerza a bajas velocidades y el coche es controlable. Y también hemos cambiado el ESC que teníamos por un driver de motor que usa PWM, así controlamos directamente el motor sin alteraciones causadas por el ESC.

## **3. Mejoras**

Además de arreglar los fallos mencionados, hemos hecho unas mejoras sustanciales respecto al año pasado.

### **3.1. SMD**

Para hacer más compacto todo el vehículo y para quitar la mayoría de cables que teníamos, ya que pueden llegar a hacer mal contacto y no son bonitos, hemos decidido hacer algo que apenas habíamos hecho antes, montar todos los componentes posibles en SMD y los que no soldarlos en placa. El montaje en SMD conlleva que los chips y microcontroladores están soldados superficialmente en la placa con patillas muy pequeñas, esto hace que quede una placa prácticamente plana y limpia de cables.

Para esto hemos necesitado un microscopio digital ya que sino no conseguimos ver para poner los componentes y soldarlos.

### 3.2. Encoders

Los encoders que usábamos el año pasado eran magnéticos y solo daban 3 pulsos por vuelta, ya que estábamos muy limitados en cuanto a espacio para montarlos. Pero hemos conseguido poner unos encoders ópticos, con unas ruedas de engranajes impresas en 3D a medida aprovechando el espacio al máximo, que dan 28 pulsos por vuelta aumentando la precisión de la lectura de distancia mucho, solo teniendo un error de  $\pm 0,3$  mm.

Para esto se ha hecho una PCB a mano con los encoders y un circuito *schmitt trigger* que convierte la salida analógica del encoder óptico a una digital al estar en un rango de voltaje. Además hemos puesto un encoder magnético directo al motor para saber su velocidad real y poder usar un controlador PID para que realmente vaya a una velocidad exacta.

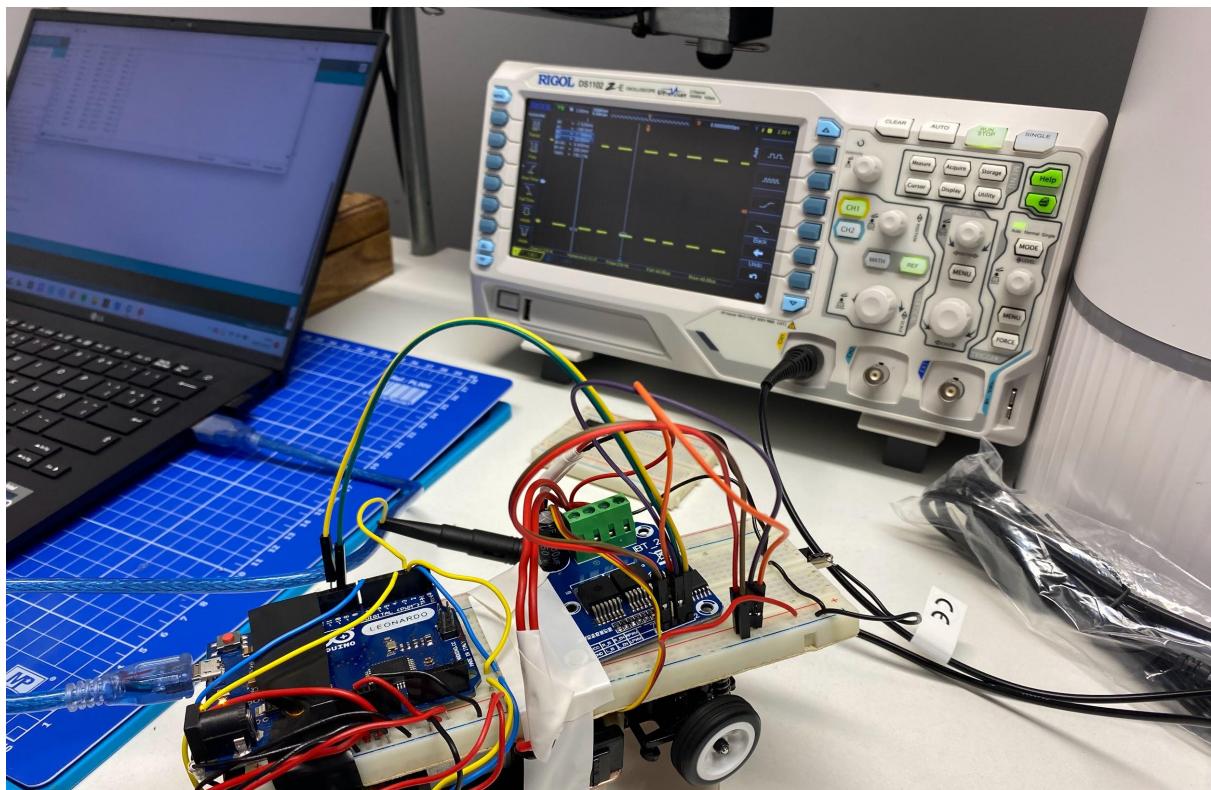


Figura 1. Comprobación de los encoders con un osciloscopio

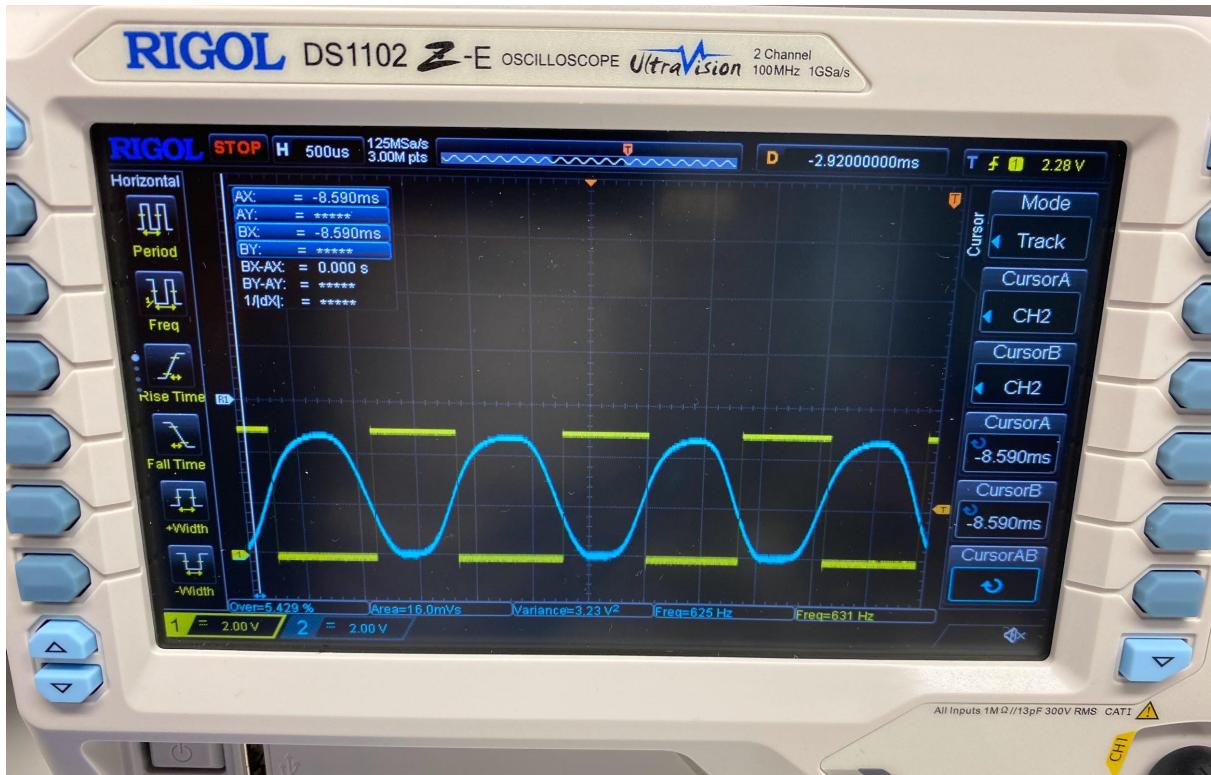


Figura 2. Pulsos de los encoders en un osciloscopio. En azul la salida directa y en amarillo la salida del *schmitt trigger*.

## 4. Programación

### 4.1. Sensores

Todos los sensores que tenemos en conjunto son: un giroscopio MPU6050, dos TOF VL53L1X, tres ultrasónicos HC-SR04 y la cámara HuskyLens.

La programación de cada uno es lo mínimo para poder leer sus respectivos datos de forma correcta, pero también hay filtros para asegurar que no hay errores. El más usado es la mediana, que es coger un número de valores, ordenarlos de menor a mayor (o viceversa) y coger el valor central, así se da como válido el valor que más aparece.

En el caso de los ultrasónicos, su programa permite activarlos o desactivarlos individualmente ya que retrasan bastante el programa.

Y el de la cámara también tiene funciones para retornar el número correspondiente (que nosotros hemos asignado) a la distribución de las señales en el sector que detecta.

## 4.2. Microcontroladores 32U4

Usamos tres microcontroladores ATMEGA32U4 para dividir el trabajo y la memoria que tienen que usar. Por esto cada uno tiene un programa específico.

Antes de poder programarlos hay que flashear un bootloader que nos permita programarlos por usb. Nosotros pusimos el del Arduino Leonardo porque lo conocemos bastante y usa este chip. Una vez tiene el bootloader, se puede programar como un Arduino Leonardo normal.

Hemos nombrado a los 32U4 como Slave de Sensores, Slave de I2C y el Master según su trabajo.

El Slave de Sensores tiene el motor, los ultrasónicos y los encoders, en general todo lo que usa interrupciones externas para funcionar. Su programa guarda en una estructura los datos de sus sensores y luego manda esta estructura al Master por SPI, también recibe comandos para encender el motor o mover el vehículo una cierta distancia con aceleraciones.

El Slave de I2C tiene los sensores que usan I2C: el MPU6050, la cámara y los TOF. Su programa también guarda los datos de los sensores en una estructura que luego manda al Master por SPI, además puede mandar directamente la distribución exacta de las señales de un sector (realmente envía un número que hemos asignado a dicha distribución). También recibe comandos del Master para encender 10 LEDs a modo de intermitentes, luz de freno y luz frontal (con largas y cortas).

El Master recibe por SPI la información de los sensores y envía comandos a ambos Slaves para mover el coche o cambiar algunos ajustes de los sensores o del programa. Con esta información se encarga de ejecutar la programación del reto propiamente dicho: girar cuando detecta que está en la esquina, girar para esquivar las señales por el lado correcto, acelerar cuando está en una recta sin nada, frenar cuando llega a una esquina o tiene que esquivar alguna señal, entre otros.

## 4.3. SPI

La comunicación SPI (*Serial Peripheral Interface*) es un protocolo de comunicación muy usado por su alta velocidad y por el hecho de que tiene comunicación *full duplex*, que quiere decir que envía y recibe información en el mismo instante de reloj. En el caso de Arduino se suele usar para recibir información de un sensor o enviarla a una pantalla OLED por ejemplo, pero nosotros lo queríamos usar para comunicar entre los microcontroladores.

Solo hay un “pequeño” problema, apenas hay documentación sobre cómo usarlo de Arduino a Arduino, de hecho la propia librería de SPI de Arduino no tiene funciones para el que

actúa de Slave, así que nos hemos tenido que buscar la vida para esto. Dimos con el siguiente foro <http://www.gammon.com.au/spi> en el que se habla de la comunicación SPI entre dos Arduinos (como Master y Slave) y vimos que el método transfer() propio de la librería SPI de arduino era igual de válido tanto para el envío de datos del Master como para la contestación y espera del Slave, lo que nos permitía recibir y enviar los datos en el Slave con una sola llamada a la interrupción (y no una por cada byte como se encuentra en el resto de ejemplos). A partir de esta premisa, profundizamos en dicha función de la librería y la modificamos en nuestro código de manera que tiene en cuenta si la línea de Slave Select está en bajo (si está en 0 indica que el Master está transmitiendo datos) siempre que espera un nuevo byte, de lo contrario descarta el envío de la estructura porque se presupone que se ha perdido información. Tras 2 semanas de investigación sobre este protocolo hemos conseguido realizar esta comunicación *full duplex* de una manera muy estable, al contrario que en los foros donde solo se envían datos al Slave sin tener en cuenta lo que éste devuelve.

## 5. La placa base

Este año hemos optado por usar la PCB como la base del coche en vez de la que tenía de aluminio, de esta forma es más compacto y estético. Para que se pueda hacer esto la placa tiene que ser lo más fina posible para no rozar el suelo, por esto y porque es algo nuevo que hemos querido probar (y porque queda bonita y profesional) hemos decidido hacerla en SMD.

### 5.1. Selección de componentes

Ya que la hemos hecho en SMD y es nuestra primera vez, hemos tenido que comprar todos los componentes (resistencias, condensadores, LEDs, etc.) en su versión SMD. Para ser capaces de soldarlos y verlos elegimos el tamaño estándar imperial 1206 (0,12 x 0,06 inch / 3,2 x 1,6 mm) ya que es mayor al usado normalmente y se puede pasar una pista de cobre entre sus pads.

El paquete de los ATMEGA32U4 que hemos usado es el TQFP-44 que es el mismo que tiene el Arduino Leonardo y es más grande y fácil de soldar que el del Pro Micro.

Y el reloj que hemos elegido tiene las mismas características que el del Leonardo con los mismos condensadores de 22pF.

## 5.2. Diseño

La parte más complicada, en nuestra opinión, es la del diseño de la PCB. A esto se le suma el hecho de que es nuestra primera vez haciendo una placa en SMD y con dos caras (porque con una sería prácticamente imposible).

Para facilitarnos este trabajo, hemos cambiado de programa de diseño de PCBs. Antes usábamos Fritzing por su simplicidad, pero hemos decidido probar EasyEda ya que su curva de dificultad es muy parecida y facilita la conexión de las pistas.

## 5.3. Creación de la PCB

### 5.3.1. Proceso de insolación y ácidos

Una vez diseñada en digital, hay que hacerla físicamente. Para esto, imprimimos en papel transparente la capa que queremos hacer. Luego este papel se pone sobre una PCB insolable (usamos unas de fibra de vidrio que conocemos ya) y se pone a unos focos durante 5 minutos para que la luz insole donde no hay pista de cobre en el diseño.

La placa ya insolada se pone en una disolución de sosa cáustica para revelarla. Y, cuando ya está revelada lo suficiente pero sin quitar las pistas, se pone en una disolución de sulfumán, agua oxigenada 110 vol. y agua que actúa como ácido, reaccionando y quitando el cobre que ha sido revelado.

De esta forma quedan solamente las pistas de cobre que hacen las conexiones.

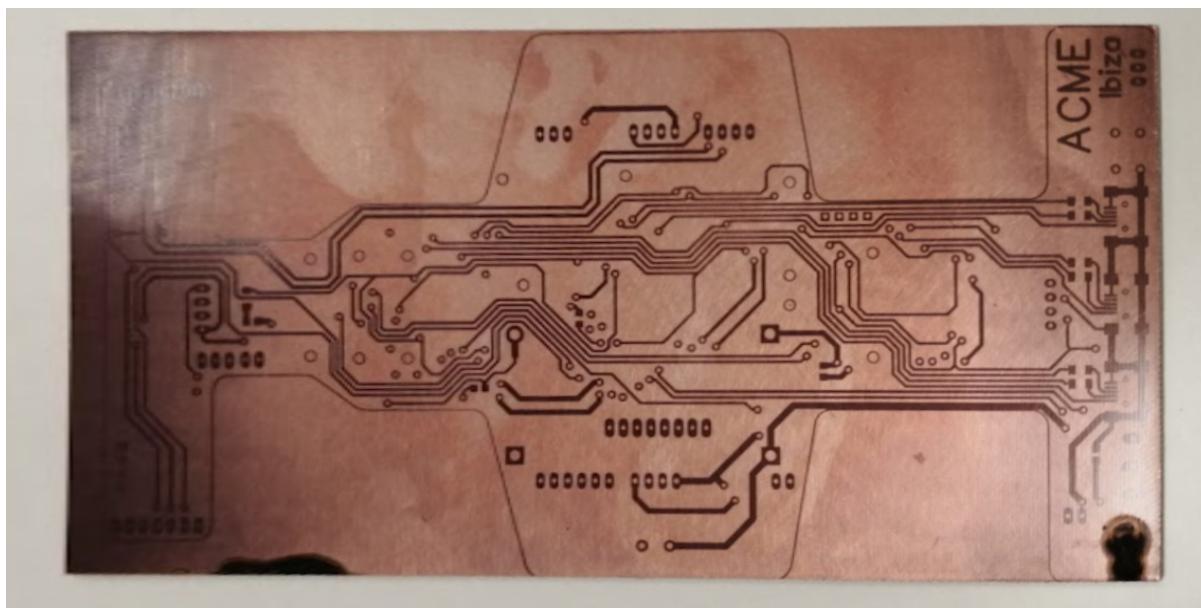


Figura 3. PCB revelada con sosa cáustica.

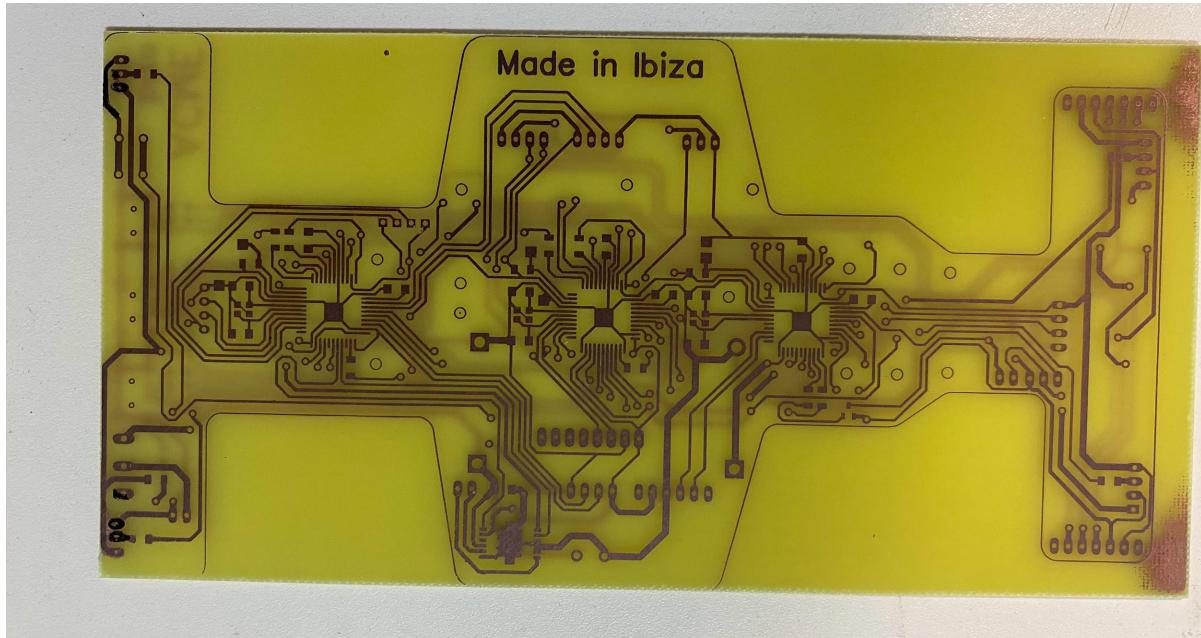


Figura 4. PCB pasada por el ácido.

### 5.3.2. Máscara de soldadura

Además de hacerla en SMD hemos querido probar otra cosa nueva, ponerle máscara de soldadura. Esta máscara es una especie de resina que cura con luz ultravioleta y protege las pistas de la oxidación, además de darle un color y diseño muy bonito y profesional a la PCB. También facilita mucho soldar ya que el estaño solo se queda en la parte sin máscara y no se va pegando a pistas cercanas.

De hecho, este tipo de máscaras es lo que se usa en las placas base profesionales.

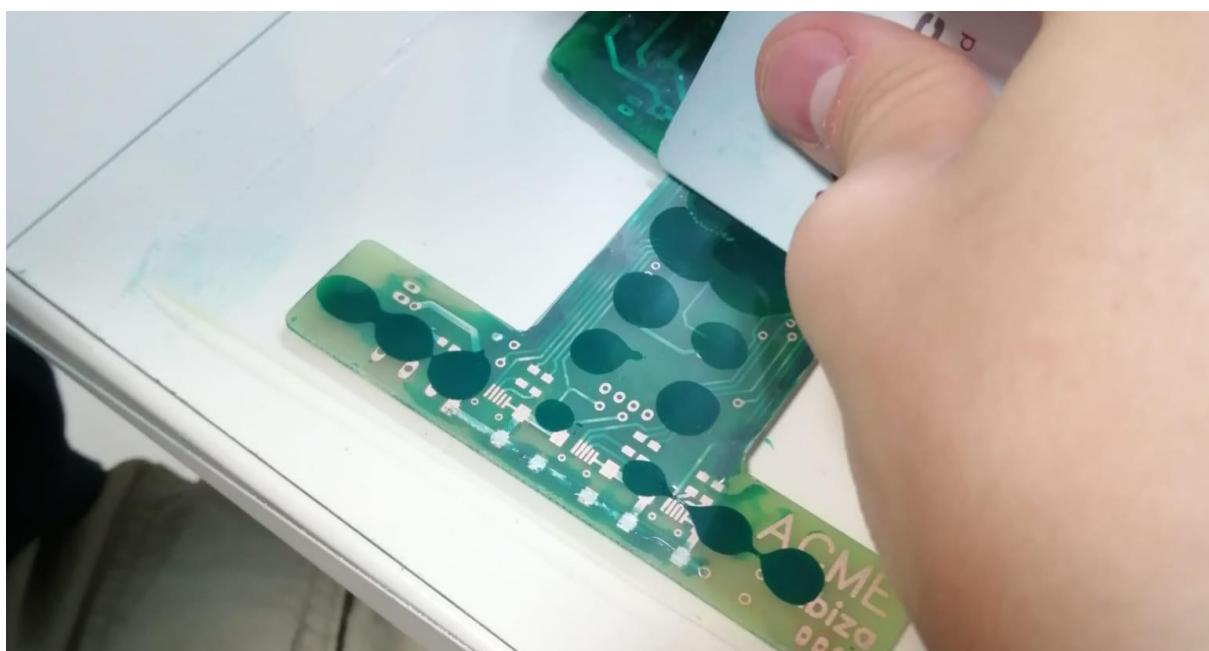


Figura 5. Aplicación de la máscara de soldadura.

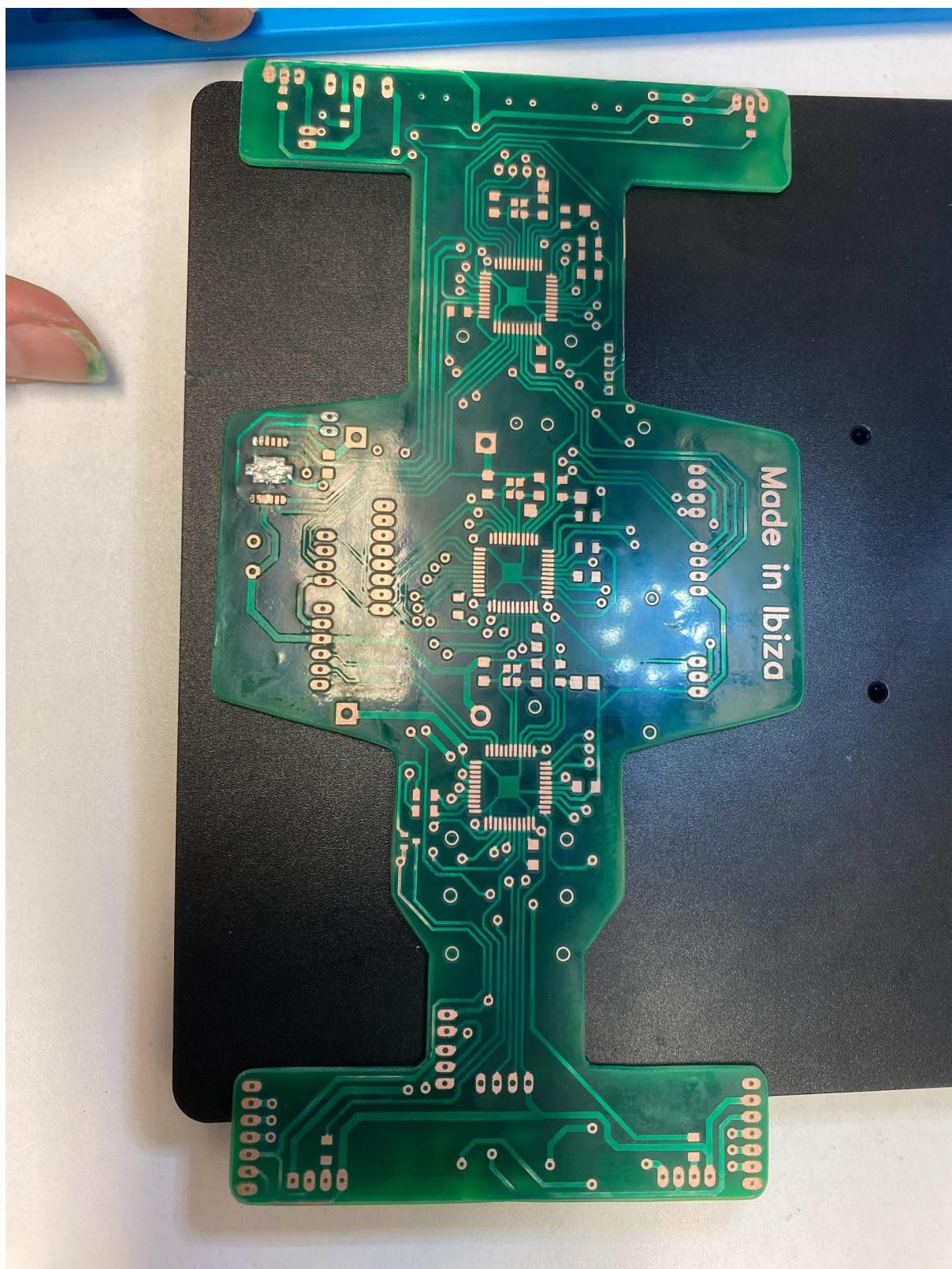


Figura 6. Máscara ya aplicada y curada.

### 5.3.3. Proceso de soldado

Cuando la placa ya tiene varias capas de la máscara curadas se puede empezar a soldar componentes.

Para soldar los componentes en SMD (las resistencias, condensadores, relojes, microcontroladores, etc.) hemos usado un soplador de aire caliente y pasta de estaño.

Aplicamos el estaño usando un palillo, y para poder ver hemos necesitado un microscopio digital. Luego con el aire caliente la pasta se funde y suelda el componente con estaño, realmente es mucho más fácil de lo que parece y esperábamos.

Después de soldar todo lo que está en SMD, había que soldar lo que va a través de agujeros: el MPU6050, los TOF, conectores macho, las placas de LEDs, entre otros. Y esto se suelda normal, usando un soldador y alambre de estaño.



Figura 7. Placa sin soldar vista en el microscopio.

Figura 8. Proceso de soldado de un 32U4 con aire caliente.

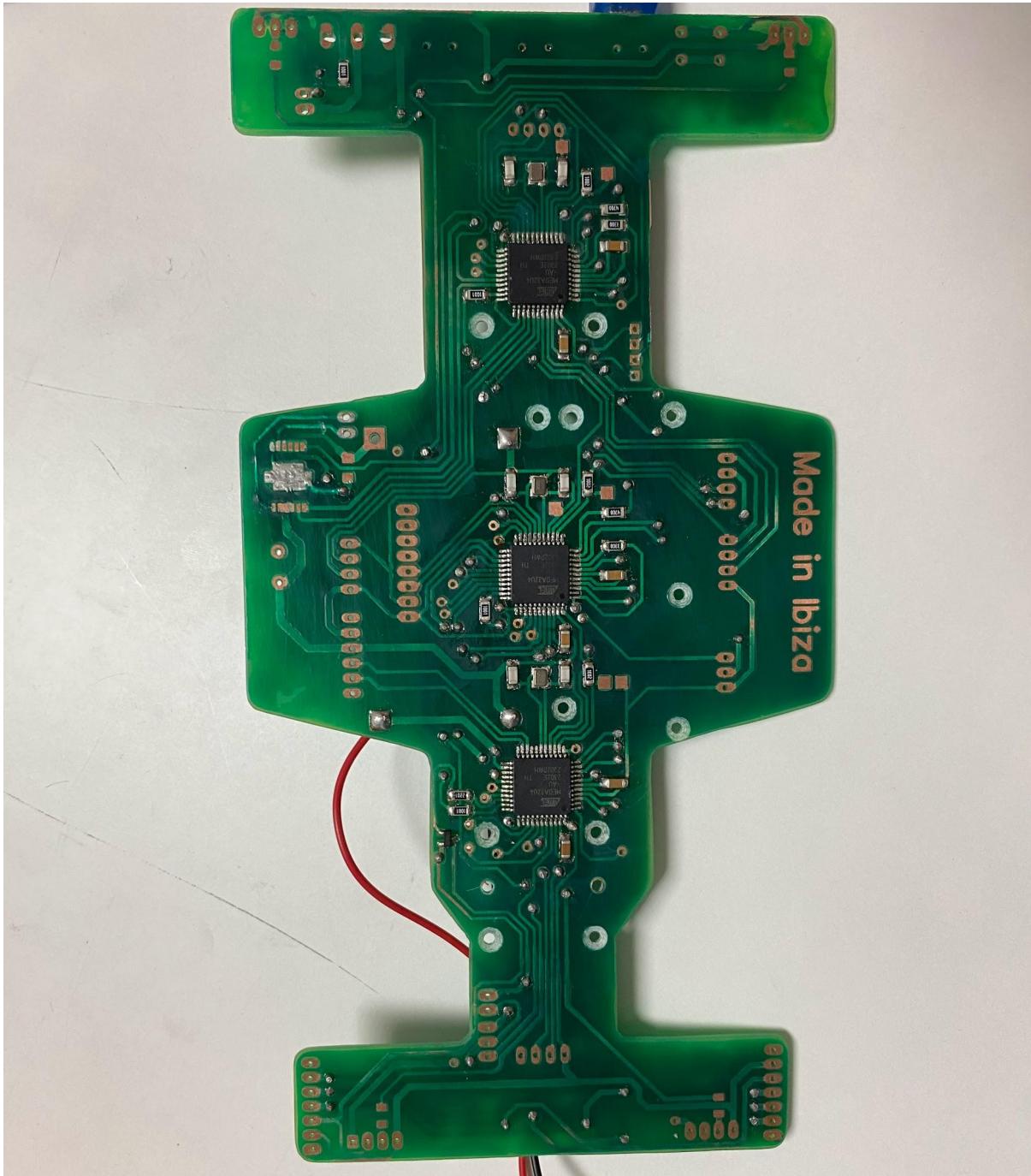


Figura 9. Placa con componentes SMD soldados.

## 6. La cámara

Para poder esquivar las señales de colores por su lado correspondiente, primero tenemos que poder saber su color. Para esto hemos montado una HuskyLens, una cámara con IA integrada que permite reconocer caras, objetos y, más importante, colores.

Para poder ver el sector y sus señales no podemos simplemente poner la cámara fija mirando hacia delante, ya que el vehículo tendría que haber girado 90º por completo y ya se

habría pasado la señal. Por eso, nuestra primera idea (y la que usamos el año pasado) era montar la cámara en un servo para girarla de izquierda a derecha, pero esto resultaba muy aparatoso y no quedaba bien.

En base a esto nos surgió otra idea, poner un espejo a  $45^\circ$  para que refleje la imagen que le llega a la cámara y así ponerla plana, mirando hacia abajo. Además este espejo se movería con un servo para tener los  $180^\circ$  de visión. Siguiendo esta idea diseñamos una carcasa para la cámara integrando un soporte para el espejo, y este soporte tiene dientes de engranaje para moverse con el engranaje del servo.



Figura 10. Renderizado digital del soporte con la cámara y el espejo.

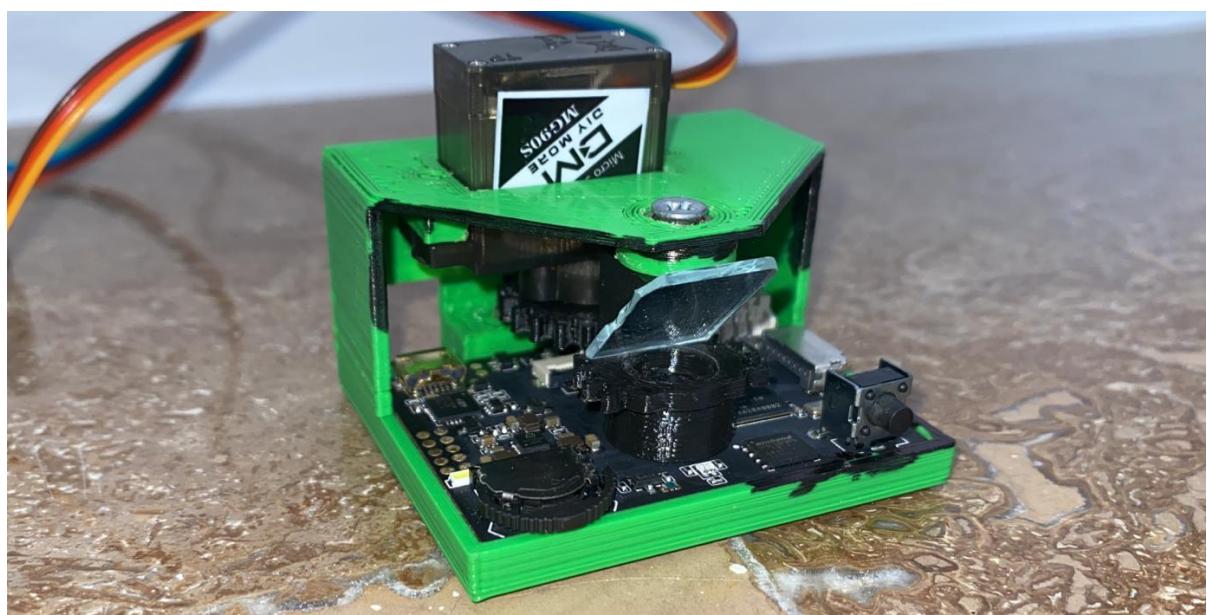


Figura 11. Montaje final de la cámara con el espejo y el servo.

En cuanto a la programación de la cámara, su API propia puede retornar la altura, anchura, posición X e Y, y su ID (que en este caso corresponde con el color que se haya registrado) de cada bloque que detecta (entiéndase bloque como cada color registrado que detecta).

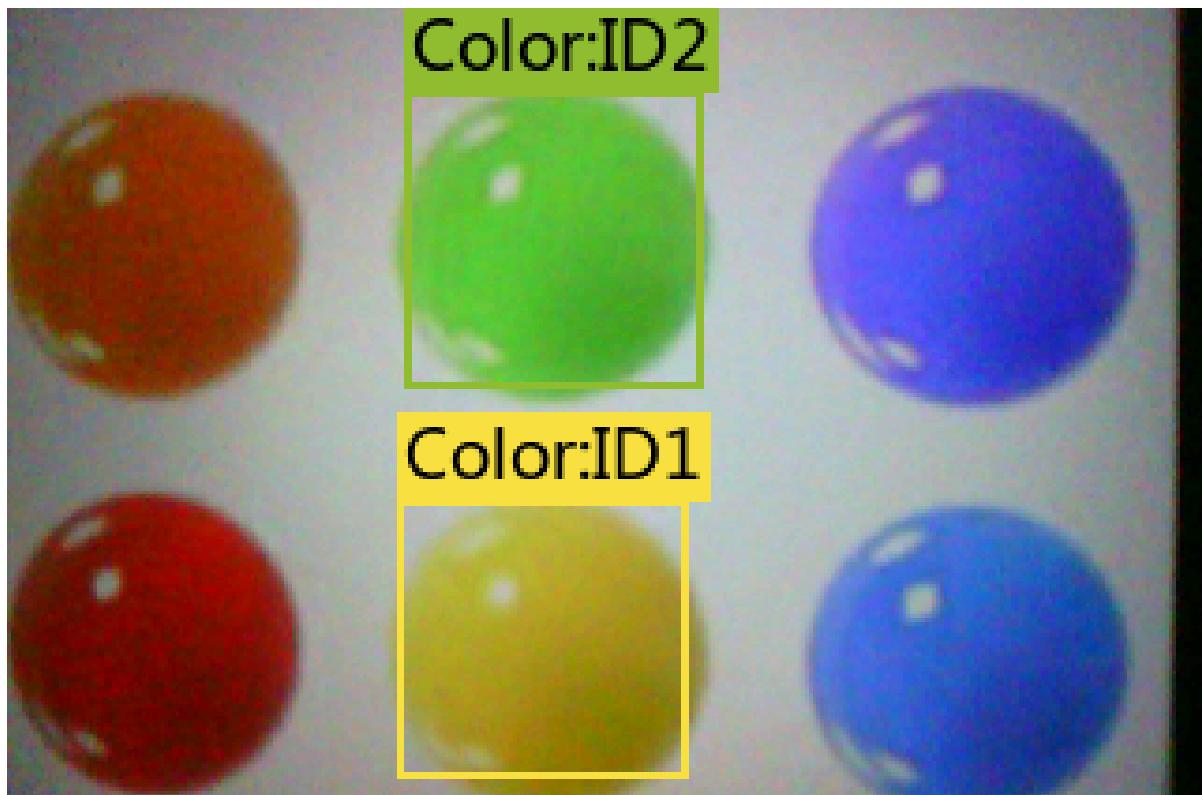


Figura 12. Ejemplo de detección de colores. Cortesía de DFRobot ([link](#)).

Con estos datos ya se puede leer la distribución de cada sector. Para esto primero se estima la distancia de cada bloque mediante una regla de 3, ya que podemos definir una constante con el tamaño real a una cierta distancia y la altura que lee la cámara a esa distancia. Y en base a esto cuanto más cerca está el objeto mayor altura lee y viceversa (este método es aproximado ya que la distorsión de la lente no es lineal, pero es un error que podemos ignorar).

Sabiendo la distancia de cada señal y su respectivo color, podríamos intentar saber cual de todas las 36 distribuciones corresponde con el sector, pero esto es completamente impráctico porque luego tendríamos que programar el coche para que pueda hacer 36 secuencias distintas dependiendo de cómo está el sector.

En vez de esto, hemos simplificado las distribuciones a 11, dividiendo el sector en posición Cerca, Medio o Lejos (ignorando si está a izquierda o derecha) ya que no nos afecta que esté en un lado u otro. Con esto sí que podemos saber a cuál de las 11 corresponde y retornar su respectivo número para saber cómo proceder.

La lectura de la posición de las señales solo se hace en la primera vuelta, de esta forma recortamos tiempo en la segunda y tercera al poder ir más rápido, con la pega de que si no lee bien en la primera vuelta fallará todo, pero como esto resultaría en el fin del intento no importa.

	Cerca	Medio	Lejos
1	V		
2	R		
3		V	
4		R	
5			V
6			R
7	V		V
8	V		R
9	R		R
10	R		V
11			

Figura 13. Tabla con los números respectivos de cada distribución.

## 7. Canal de YouTube y videos

En nuestro canal de YouTube ([link](#)) se pueden encontrar videos que vamos subiendo. En especial los dos videos del vehículo haciendo los retos:

▶ Open Challenge Demo FE ACME 2023

▶ Obstacle Challenge FE ACME 2023

## 8. Debugging

Con la finalidad de encontrar errores y saber qué está haciendo realmente el vehículo teníamos que implementar un sistema para hacer debugging. Para esto le añadimos un módulo bluetooth HC-05 que envía los datos de la velocidad, encoders y ciertas flags en tiempo real.

Además usamos un programa que genera gráficas con esta información hecho por nuestra amiga María Pilligua y subido a su GitHub ([https://github.com/mpilligua/app\\_wro](https://github.com/mpilligua/app_wro)).

## 9. Fotos del vehículo final

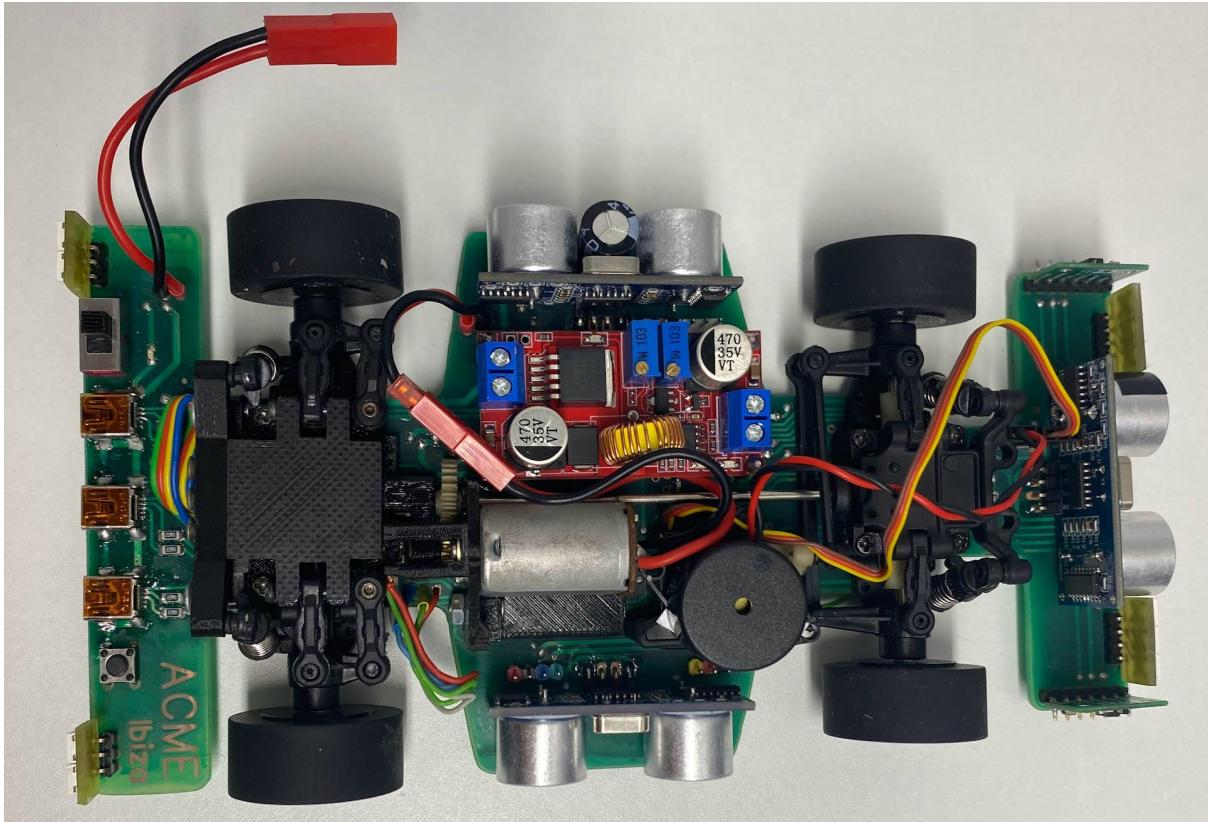


Figura 10. Vista superior del vehículo final.

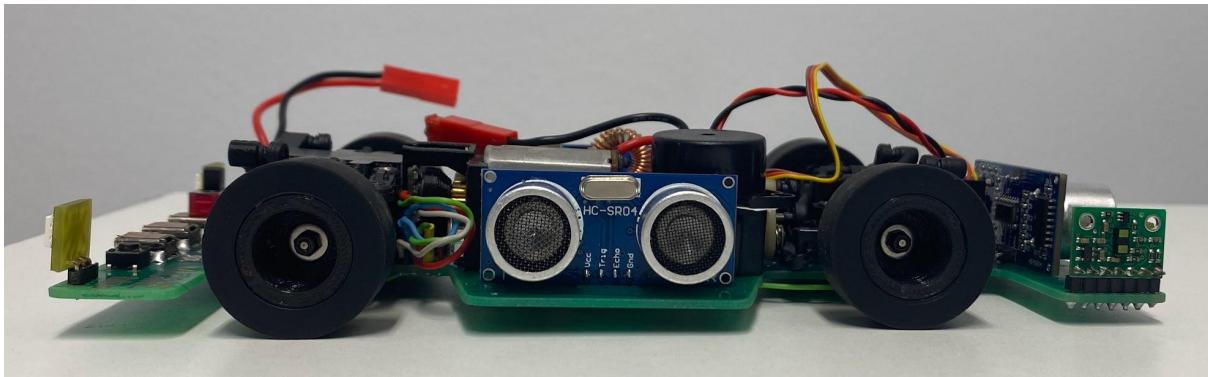


Figura 11. Vista del lateral derecho del vehículo final.

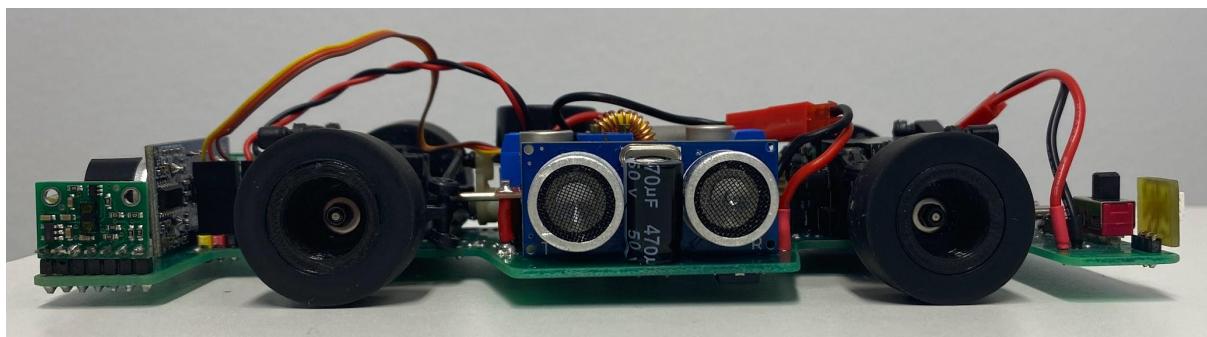


Figura 12. Vista del lateral izquierdo del vehículo final.

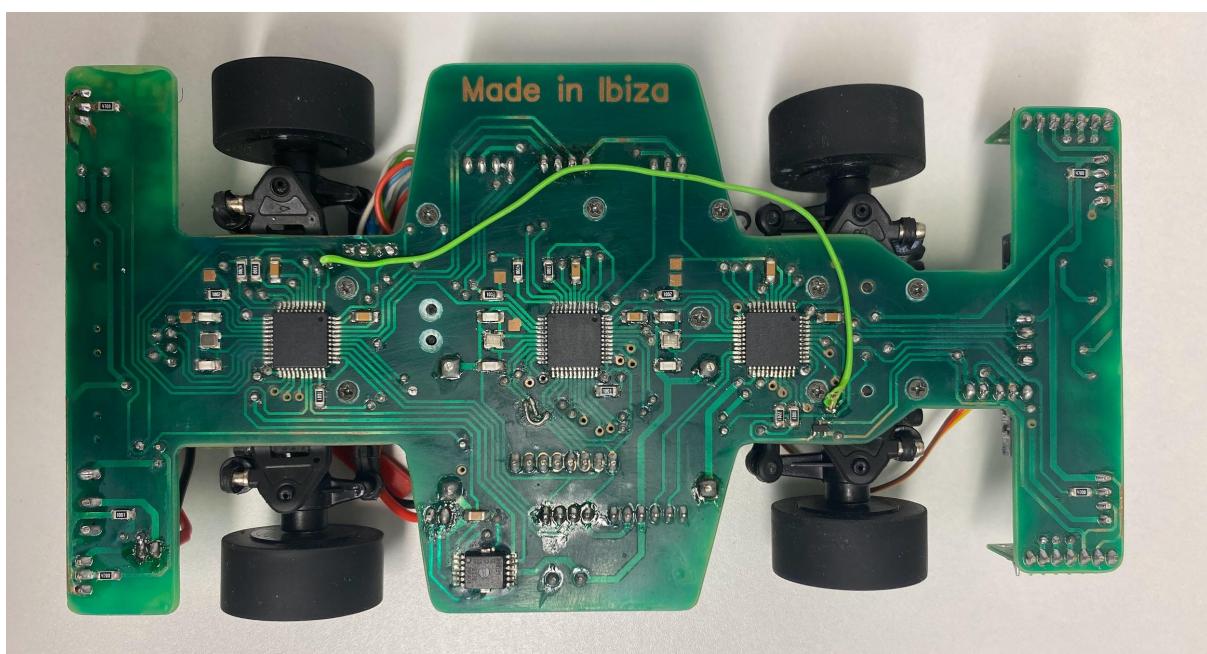


Figura 13. Vista inferior del vehículo final.

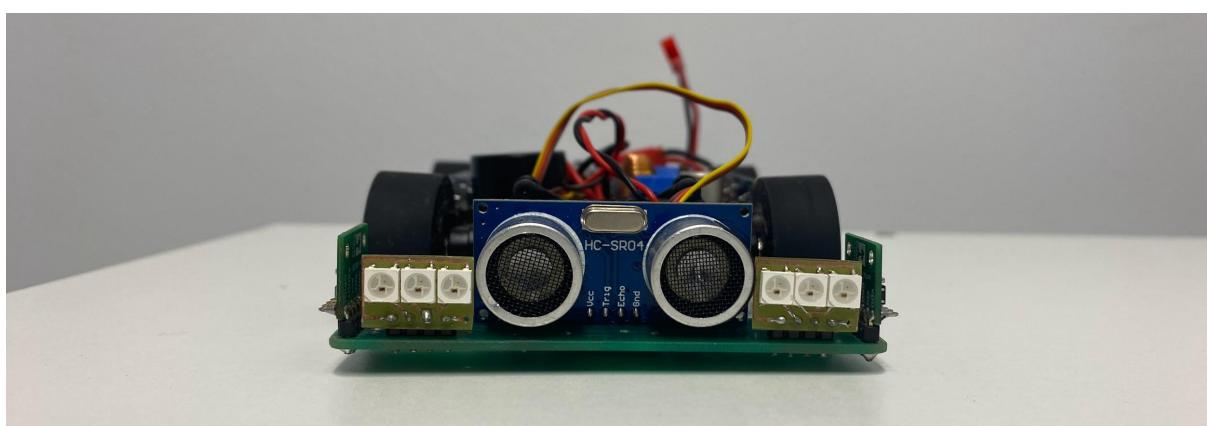


Figura 14. Vista frontal del vehículo final.

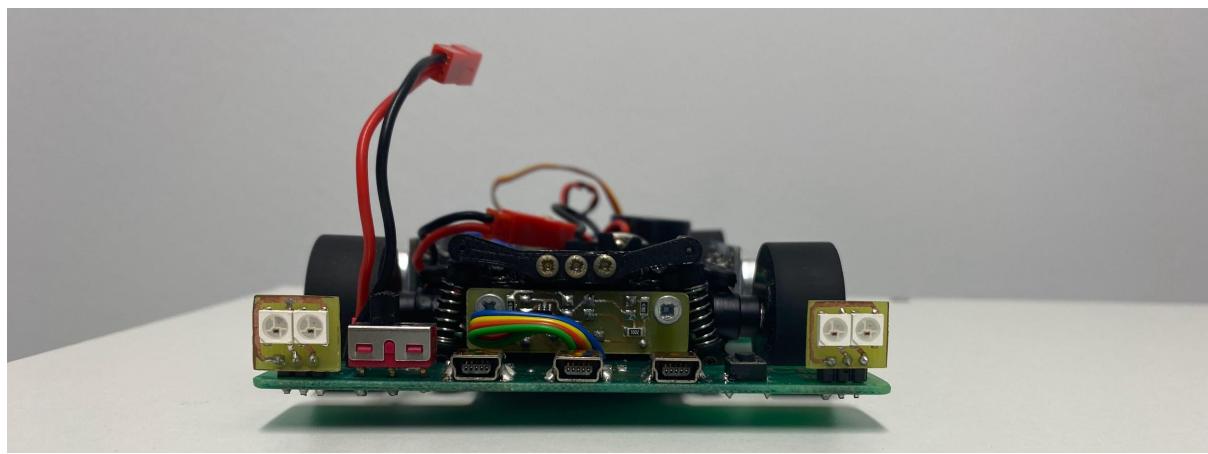


Figura 15. Vista trasera del vehículo final.