ACME Future Engineers 2023 Engineering Journal

Table of contents

1. Introduction

We are the ACME team, participants in the WRO Future Engineers 2023.
This document is intended to show the process followed to create our vehicle for this competition, as well as to include images of certain moments in the design and construction.

2. Last year’s bugs

In addition to participation this year, we also participated in both the previous and 2021 competitions. Since our robot is based on last year's design, we have to address the serious bugs it had.
A key issue was the substantial electrical instability that resulted from the use of two different voltages, 5V and 3.3V, which occasionally caused the system to reset. However, we had to use both, as the M5Stack microcontroller uses 3.3V and the Arduino uses 5V. This year, we decided to use 5V with the 32U4 microcontrollers as if they were standard Arduinos, and omitted the former.
Additionally, the car's speed was a drawback as it could only go too fast and lacked the ability to slow down, hindering specific maneuvers. To address this, we added a smaller gear to the engine, effectively reducing the speed to a third of its previous value, increasing power at lower speeds and allowing better control of the car. We also replaced the existing ESC with a PWM motor driver. This allows us to control the motor directly, without any changes caused by the ESC.

3. Upgrades

In addition to fixing the aforementioned bugs, we have made substantial improvements over last year.

3.1. SMD

In order to make the whole car more compact and to get rid of most of the wires that we had, as they can make bad contact and are not very pretty, we decided to do something that we had hardly ever done before, which was to mount all possible components in SMD and solder the ones that we couldn't on the board. SMD assembly means that the chips and microcontrollers are soldered to the surface of the board with very small pins, which means that the board is virtually flat and free of wires.

We needed a digital microscope for this, otherwise we would not be able to see how to place and solder the components.

3.2. Encoders

The encoders we used last year were magnetic and only gave 3 pulses per revolution because we had very limited space to mount them. But we managed to fit optical encoders, with custom 3D-printed gears to make the most of the space, which give 28 pulses per revolution, greatly improving the accuracy of the distance reading, which has an error of only ±0.3 mm.

To do this, we built a handmade PCB with the encoders and a Schmitt trigger circuit that converts the analogue output of the optical encoder to digital when it is in a voltage range.

We also fitted a magnetic encoder directly to the motor to know its real speed. This allowed us to use a PID controller to make it run at an exact speed.

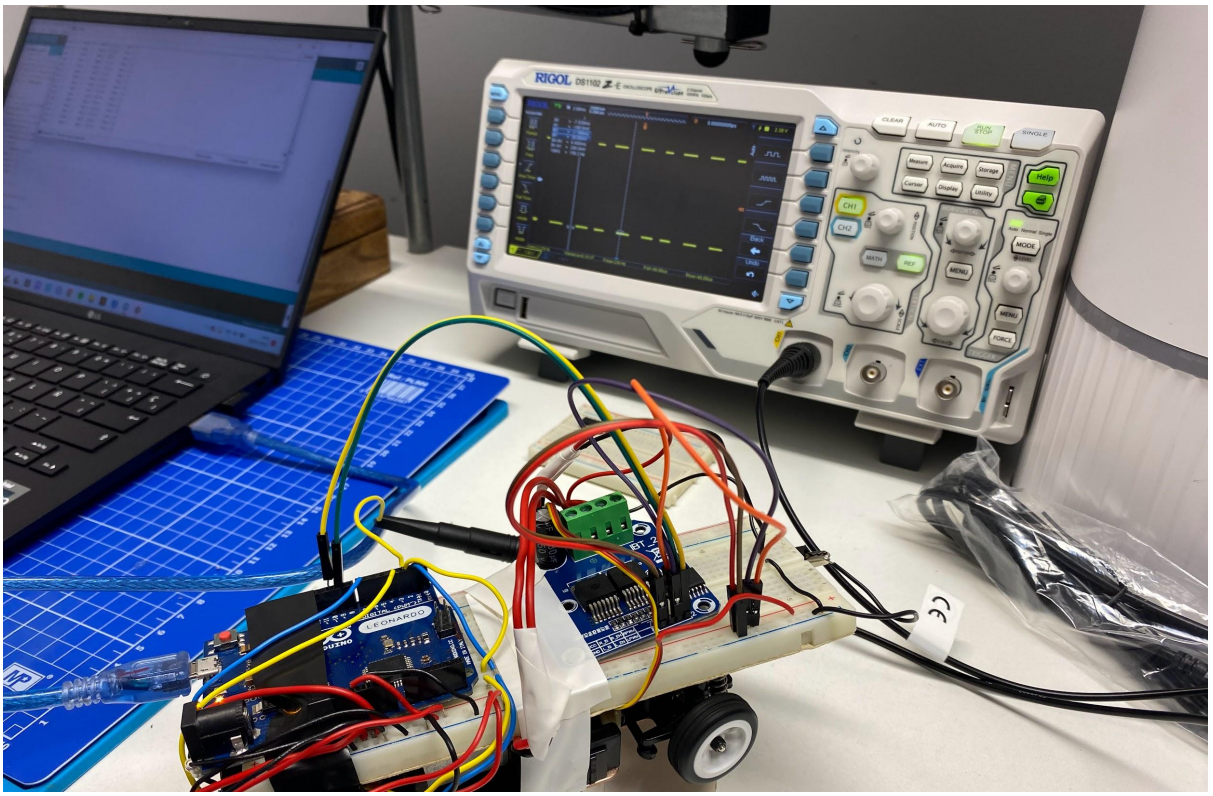

Figure 1. Encoders comprobation using an oscilloscope.

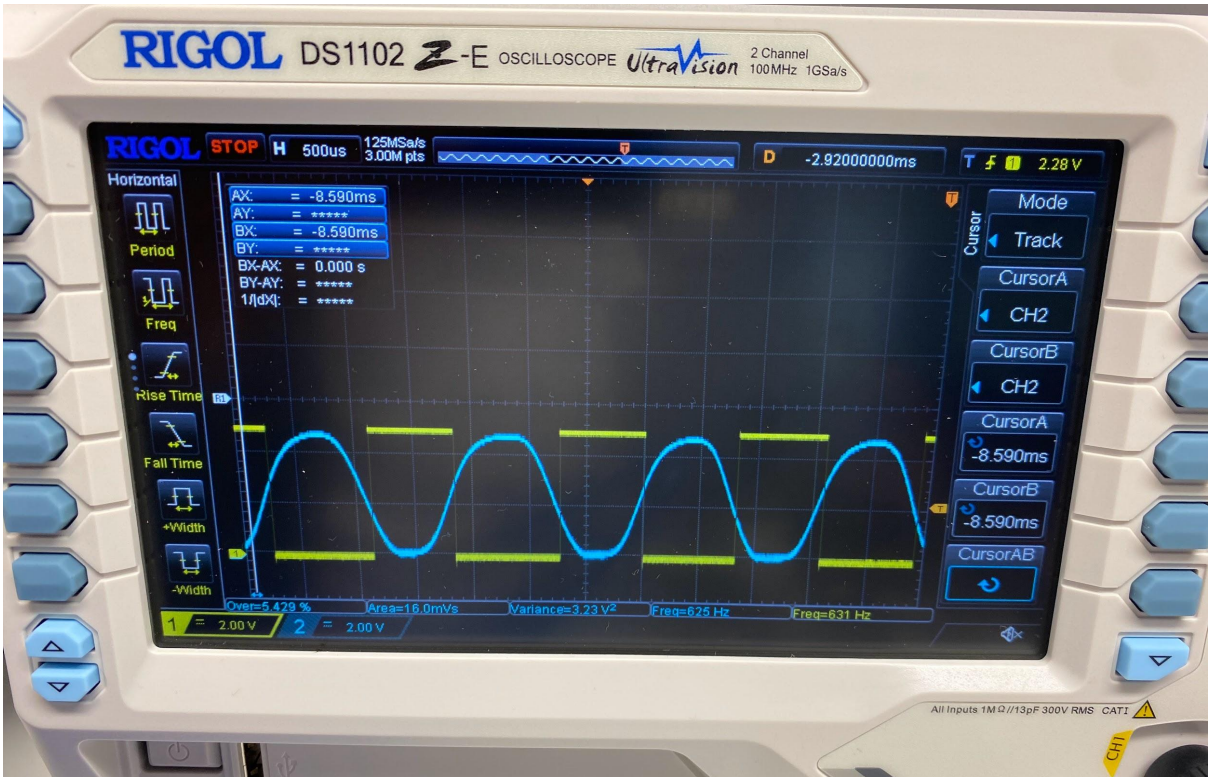

Figure 2. Encoder pulses in an oscilloscope. In blue the raw output and in yellow the Schmitt trigger output.

4. Programming

4.1. Sensors

All the sensors that we have are: an MPU6050 gyroscope, two VL53L1X TOF sensors, three HC-SR04 ultrasonic sensors and the HuskyLens camera.

The programming of each of them is the minimum required to be able to read their respective data correctly, but there are also filters to make sure that there are no errors. The most commonly used is the median, which involves taking a number of values, ordering them from smallest to largest (or vice versa) and taking the central value, so that the value that appears most often is taken as valid.

In the case of the ultrasonic sensor, the program allows you to activate or deactivate them individually, as they slow down the program considerably.

The camera also has functions for returning the number (which we have assigned) corresponding to the distribution of the signals in the sector that it detects.

4.2. 32U4 Microcontrollers

We utilize three ATMEGA32U4 microcontrollers to distribute workloads and allocate memory usage. A specific program is written for each of them.
Before programming, we need to flash a bootloader that allows USB programming. We have chosen the Arduino Leonardo bootloader due to our familiarity with it and the fact that it uses this chip. Once the bootloader is installed, the programming process is identical to that of a standard Arduino Leonardo.
The microcontrollers are designated Sensor Slave, I2C Slave and Master respectively, according to their assigned roles.
The Sensor Slave handles the motor, the ultrasonic sensor and the encoders, in general all devices that use external interrupts to work. Its programming stores sensor data in a structure and transmits it to the Master via SPI, while also receiving instructions to activate the motor or move the vehicle a certain distance with accelerations.
The I2C Slave has all the sensors that use I2C: the MPU6050, the camera and the TOFs. Its program stores the sensor data in a structure, which it then transmits to the Master through SPI. It can also send the exact signal distribution of a sector directly (sending a number that we have assigned to this distribution). It also receives instructions from the Master to activate or deactivate 10 LEDs functioning as the car's indicators, brake light and front light.
The Master receives data from the sensors via SPI and issues commands to the Slaves to alter the car's movement or adjust program settings. It uses this data to execute the challenge's programming, including turning in corners, avoiding signals on the correct side, accelerating when there is nothing on a straight line, braking when it reaches a corner or has to avoid a signal, and so on.

4.3. SPI

The SPI (Serial Peripheral Interface) protocol is often used due to its high speed and full duplex communication, which allows simultaneous transmission and reception of data. While typically used in Arduino to receive data from a sensor or send it to an OLED screen for example, we wanted to use it to communicate between microcontrollers.
There is only one "small" problem, there is hardly any documentation on how to use it between two Arduinos. Additionally, the Arduino SPI library offers no functions for the Slave variant, so we were forced to look for a way to do this. We found a forum at http://www.gammon.com.au/spi where the SPI communication between two Arduinos (as Master and Slave) is discussed. We noticed that the transfer() method of the Arduino SPI

library was suitable for sending data from the Master and for responding and waiting for the Slave. This allowed us to receive and send data on the Slave with a single call to the interrupt, rather than one for each byte as seen in other examples. Based on this premise, we delved further into the library function and modified it in our code so that it takes into account if the Slave Select line is low (which indicates the Master is transmitting data) whenever it expects a new byte, otherwise it discards the structure sent because it assumes that information has been lost. After two weeks of researching this protocol, we achieved a completely stable full-duplex and fast communication, unlike forums where data is sent to the slave regardless of what the slave sends back.

5. The motherboard

This year we have chosen to use the PCB as the base of the car instead of the aluminium one, this way it is more compact and aesthetic. To be able to do this, the board has to be as thin as possible so that it doesn't rub the ground, for this reason and because it is something new that we wanted to try (and because it looks nice and professional) we decided to make it in SMD.

5.1. Selection of components

Since this is our first time making it in SMD, we had to buy all the components (such as resistors, capacitors, and LEDs) in their SMD version. We chose the standard Imperial 1206 size (0.12 x 0.06 inches / 3.2 x 1.6 mm) for easier soldering and better visibility. It also allows a copper trace to be placed between the pads.
We used the TQFP-44 package for the ATMEGA32U4, which is identical to that of the Arduino Leonardo and notably, larger and more manageable to solder than the Pro Micro.
And the clock we have selected possesses equivalent characteristics to the Leonardo with the identical 22pF capacitors.

5.2. Design

The PCB design is the most challenging aspect of the motherboard, particularly since we are creating an SMD board with double-sided traces, as it would be virtually impossible to do it with only one side.
For ease of use, we've switched from Fritzing to EasyEda because its difficulty level is comparable and it simplifies routing the tracks.

5.3. Making the PCB

5.3.1. Insolation and acid process

Once the digital design is complete, it must be physically produced. To accomplish this, we print the desired layer onto transparent paper. The sheet is then positioned on a photosensitive Printed Circuit Board (PCB) made of fiberglass which undergoes a 5 minute exposure to the light source, excluding copper traces from the design.

Afterwards, the exposed PCB is placed in a caustic soda solution to develop it. Once sufficiently developed, but without removing the traces, it is placed in a solution of hydrochloric acid, hydrogen peroxide at 110 volumes and water. This reacts with the developed copper and removes it. This leaves only the copper tracks that make the necessary connections.

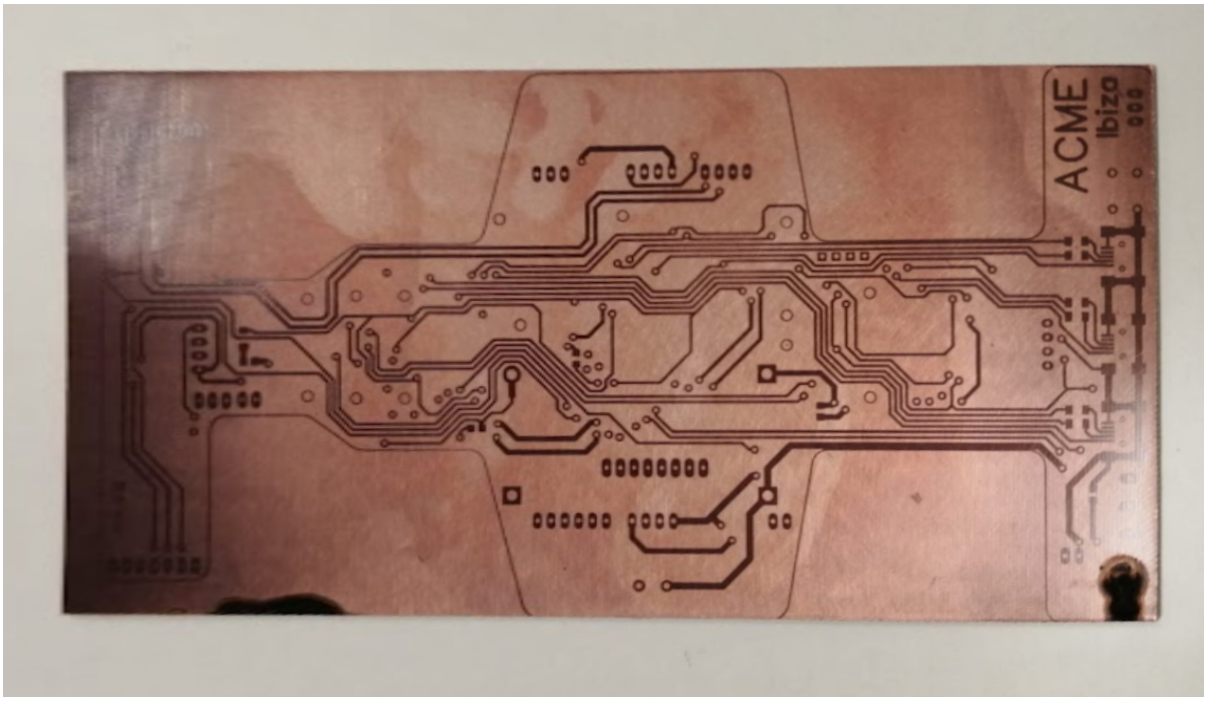

Figure 3. Fully developed PCB using caustic soda.

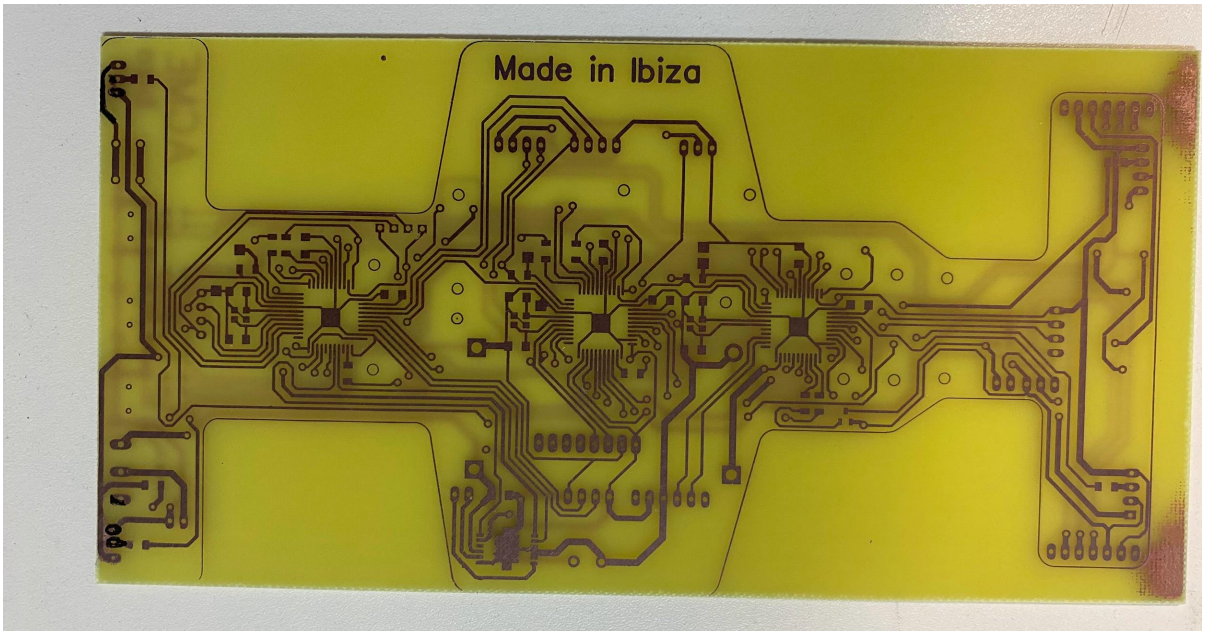

Figure 4. PCB after being treated with the acid.

5.3.2. Solder mask

In addition to SMD, we wanted to try something new and apply a solder mask. This mask, made from a resin that cures under ultraviolet light, protects the tracks from oxidation and gives the PCB a professional look with a clean colour and design. Additionally, it simplifies soldering since the solder adheres only to the unmasked portion and not to the nearby traces. It is worth noting that this form of mask is commonly seen on high-quality motherboards used in professional settings.

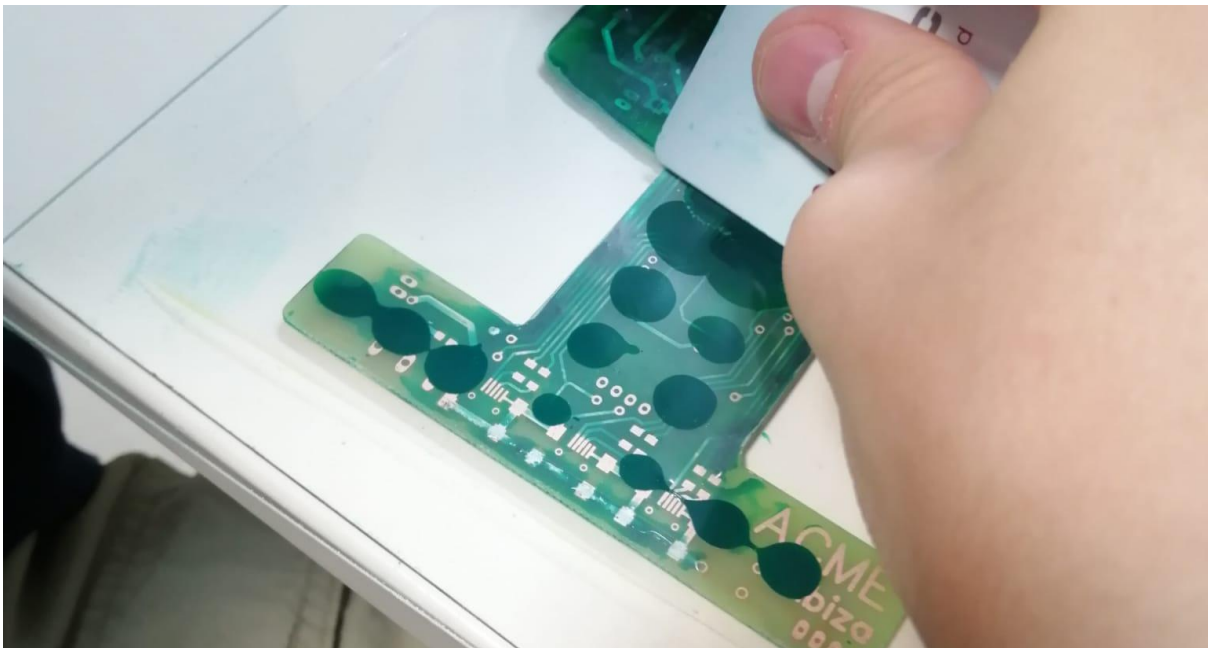

Figure 5. Solder mask application process.

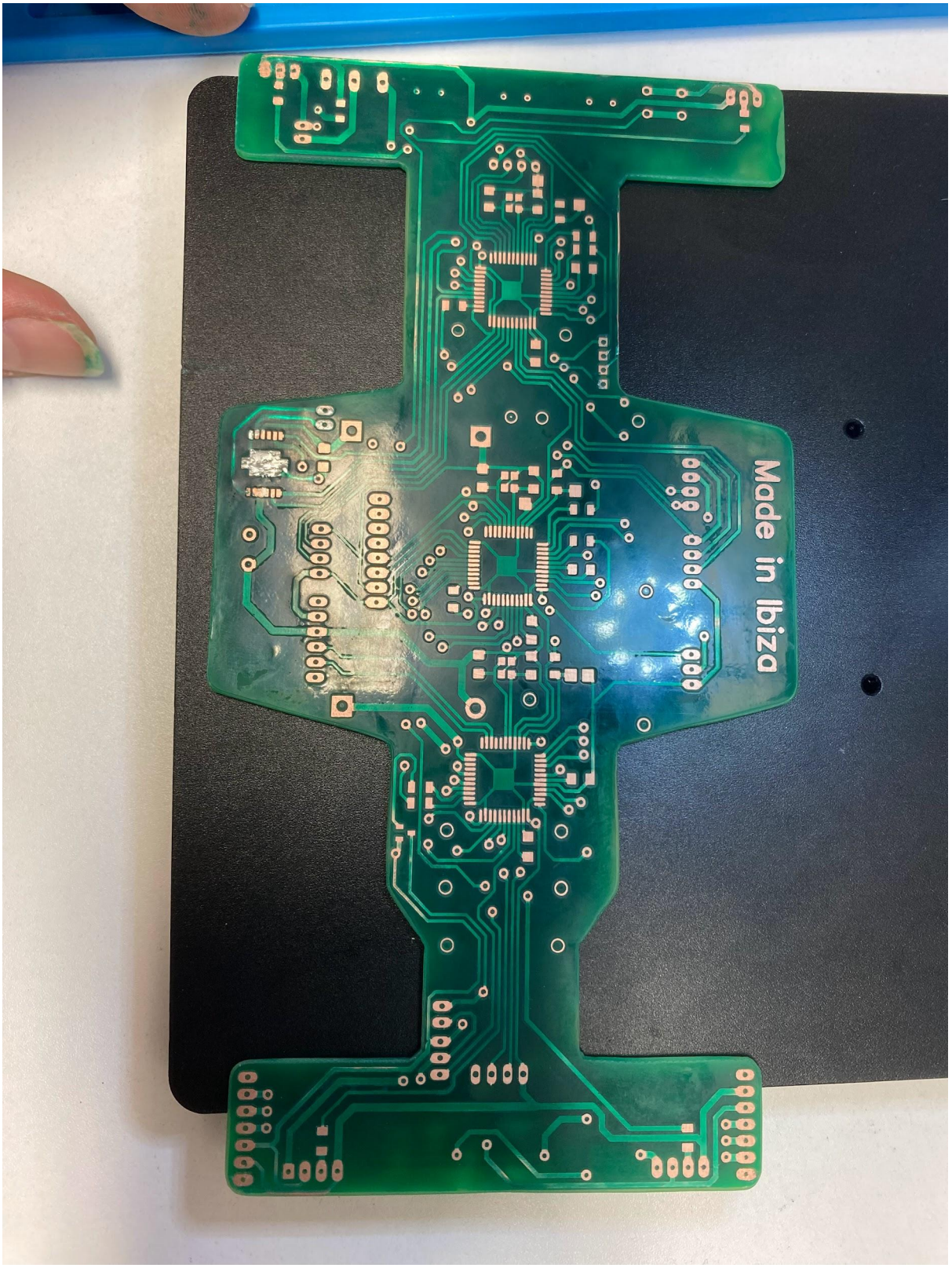

Figure 6. Fully applied and cured solder mask.

5.3.3. Soldering process

Once the board has cured several layers of the mask, the components can be soldered.

We used a hot air blower and tin paste for SMD components such as resistors, capacitors, clocks and microcontrollers.

We applied the tin with a toothpick and used a digital microscope to improve visibility. Subsequently, activating the hot air melts the paste and solders the component with tin, which was easier than we expected.

After soldering all SMD components, we proceeded to solder through-hole components such as the MPU6050, TOFs, male connectors, and LED boards using a soldering iron and tin wire.

Figure 7. Not soldered PCB viewed in the microscope.

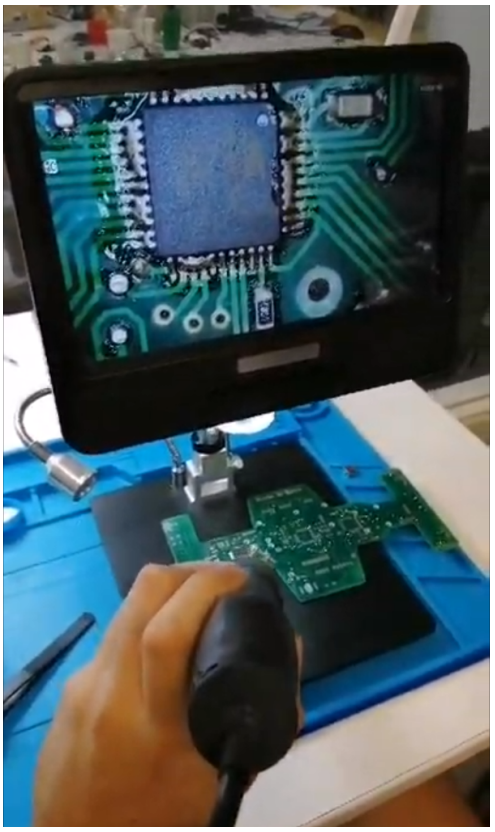

Figure 8. Soldering process of a 32U4 with hot air.

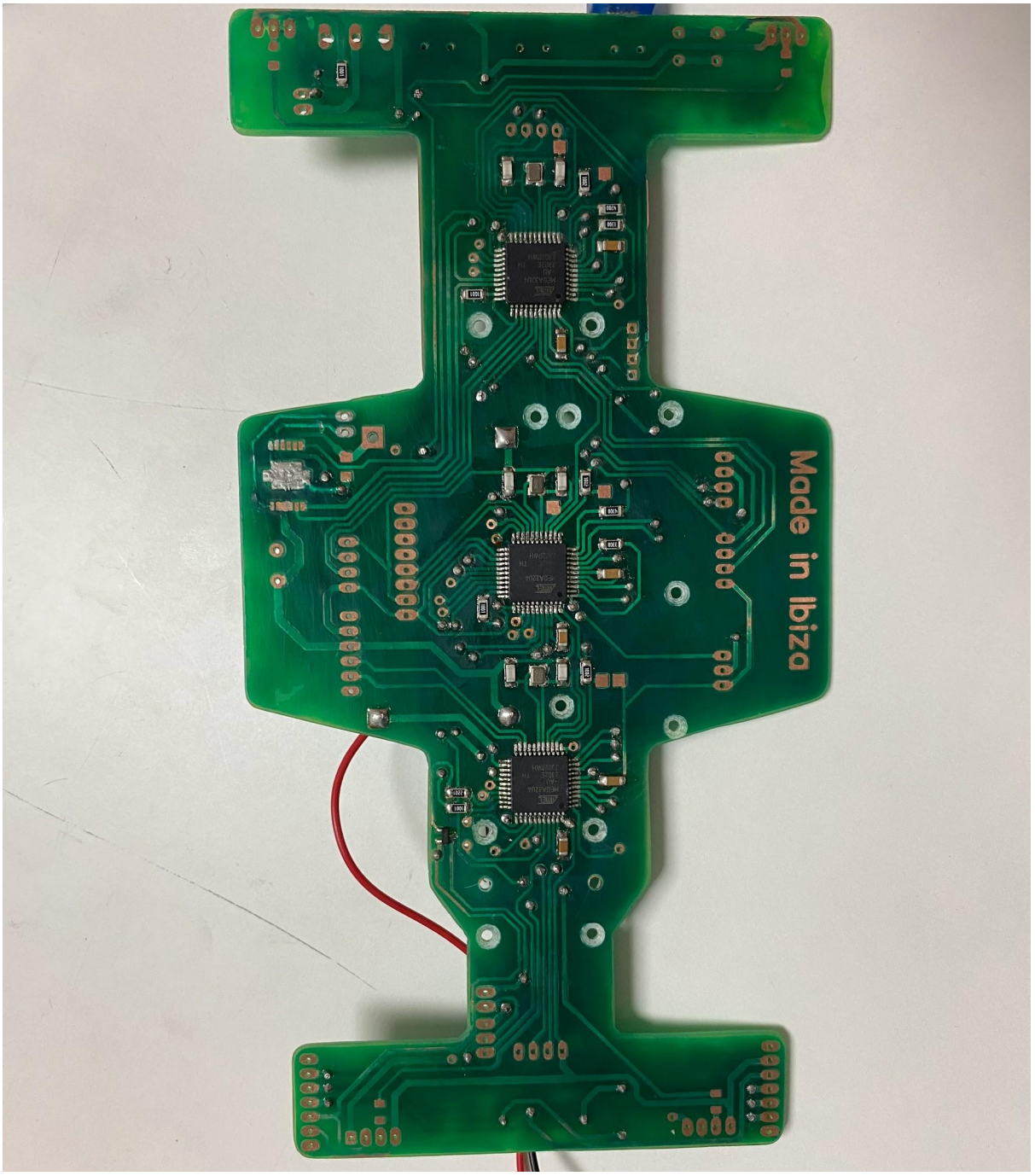

Figure 9. Board with SMD components soldered.

6. The camera

In order for us to be able to avoid coloured signs on the appropriate side, we first need to be able to recognise the colour of the sign. To do this, we mounted a HuskyLens, a camera with built-in AI that can recognise faces, objects and, most importantly, colours.

To see the sector and its signals, we cannot just fix the camera to the front, as the vehicle would have to turn a full 90° and the signal would have already passed. Therefore, our first

idea (and the one we used last year) was to mount the camera on a servo to turn it from the left to the right, but this was too bulky and did not look good.

We then came up with another idea, which was to place a mirror at 45º to reflect the image coming into the camera, so that the camera could face downwards. In addition, this mirror would move with a servo to have a 180º view. Following this idea, we designed a case for the camera, integrating a support for the mirror, and this support has gear teeth so that it can move with the servo gear.

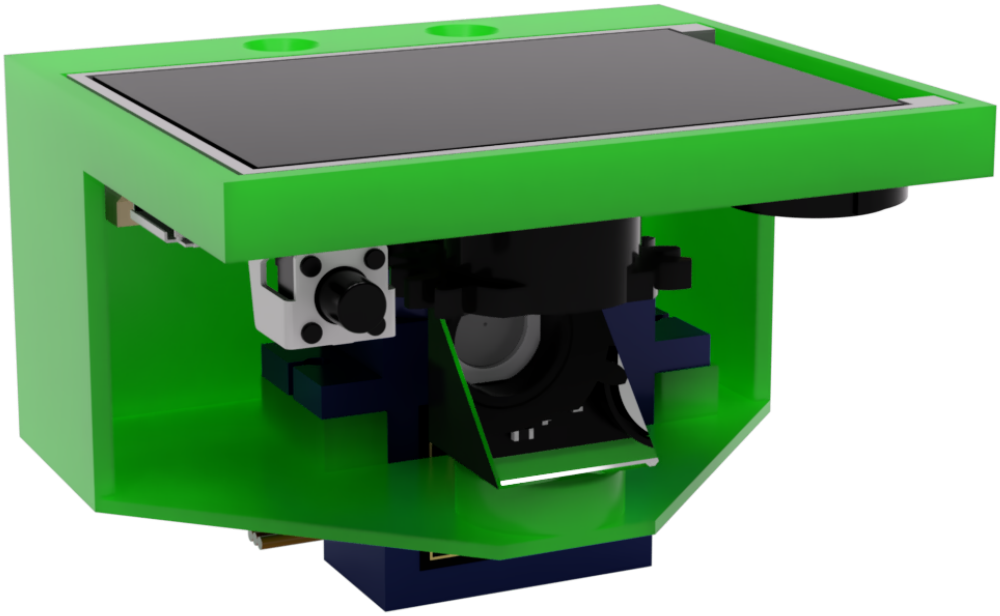

Figure 10. Digital rendering of the camera's case with the mirror.

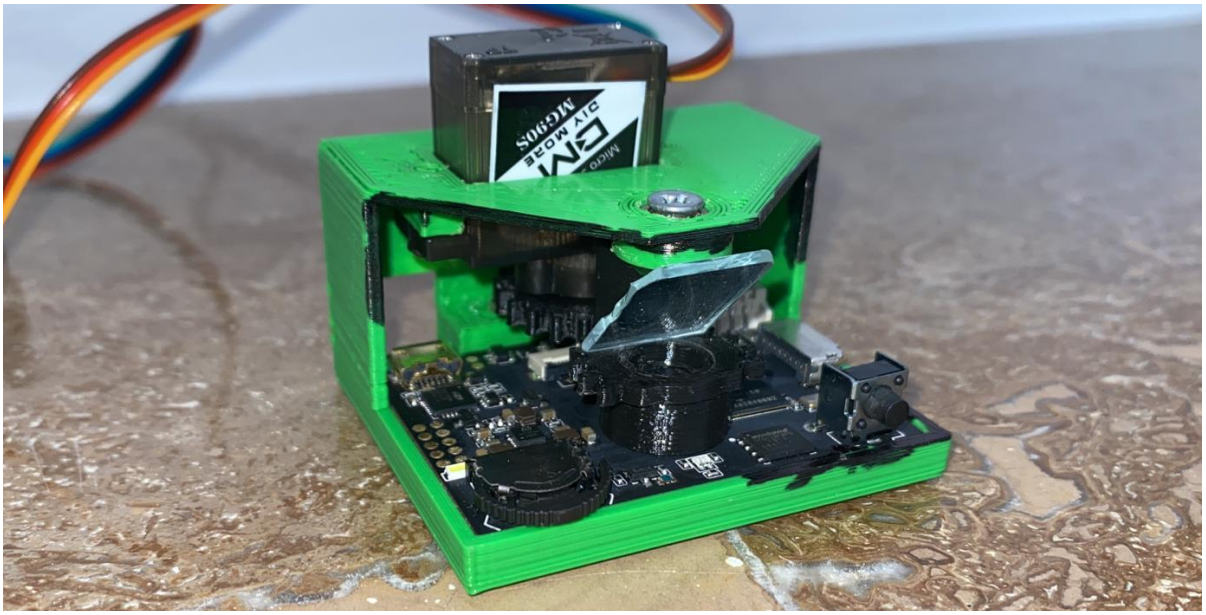

Figure 11. Final assembly of the camera, mirror and servo.

As for the programming of the camera, the API can retrieve the height, the width, the X and Y position and the ID of each block that it detects. In this case, the ID corresponds to the colour that is registered.

Figure 12. Colour detection example. Courtesy of DFRobot (link).

By using this data, it is possible to read the distribution of each sector. To begin, we estimate the distance of each block using a rule of three. This involves setting a constant, based on the real size of the object at a certain distance and the height that the camera reads at that distance. As the object gets closer, its height when read by the camera increases, and vice versa. It is important to note that this method is only approximate due to the distortion of the lens not being linear, however we can disregard this error for our purposes.

Knowing the distance and colour of each signal, we could attempt to determine which of the 36 distributions corresponds to the sector. However, this approach is impractical as it would require programming the car to execute 36 different sequences depending on the sector.

Instead, we have simplified the distributions into 11 possibilities, dividing the sector into Near, Middle or Far position, regardless of whether it is on the left or right. We can then determine the corresponding distribution and return its respective number to know how to proceed.

The signal's position reading occurs solely during the first lap to save time during subsequent laps, allowing for greater speed. This comes with the disadvantage, however,

that if the reading fails during the first lap, the entire attempt fails. Nonetheless, such an occurrence holds no consequence in the end.

	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			

Figure 13. Table showing the respective numbers for each distribution.

7. YouTube channel and videos

On our YouTube channel (link), you can find uploaded videos, including two clips of the vehicle attempting the challenges:

Open Challenge Demo FE ACME 2023

Obstacle Challenge FE ACME 2023

8. Debugging

In order to identify errors accurately and obtain reliable vehicle performance data, we implemented a debugging system. To achieve this, we integrated a Bluetooth HC-05 module that transmits real-time information about the speed, encoders, and specific flags. Additionally, we utilized a program created by our friend María Pilligua, which generates graphs based on this information. The program was uploaded to her GitHub repository (https://github.com/mpilligua/app_wro).

9. Final vehicle photos

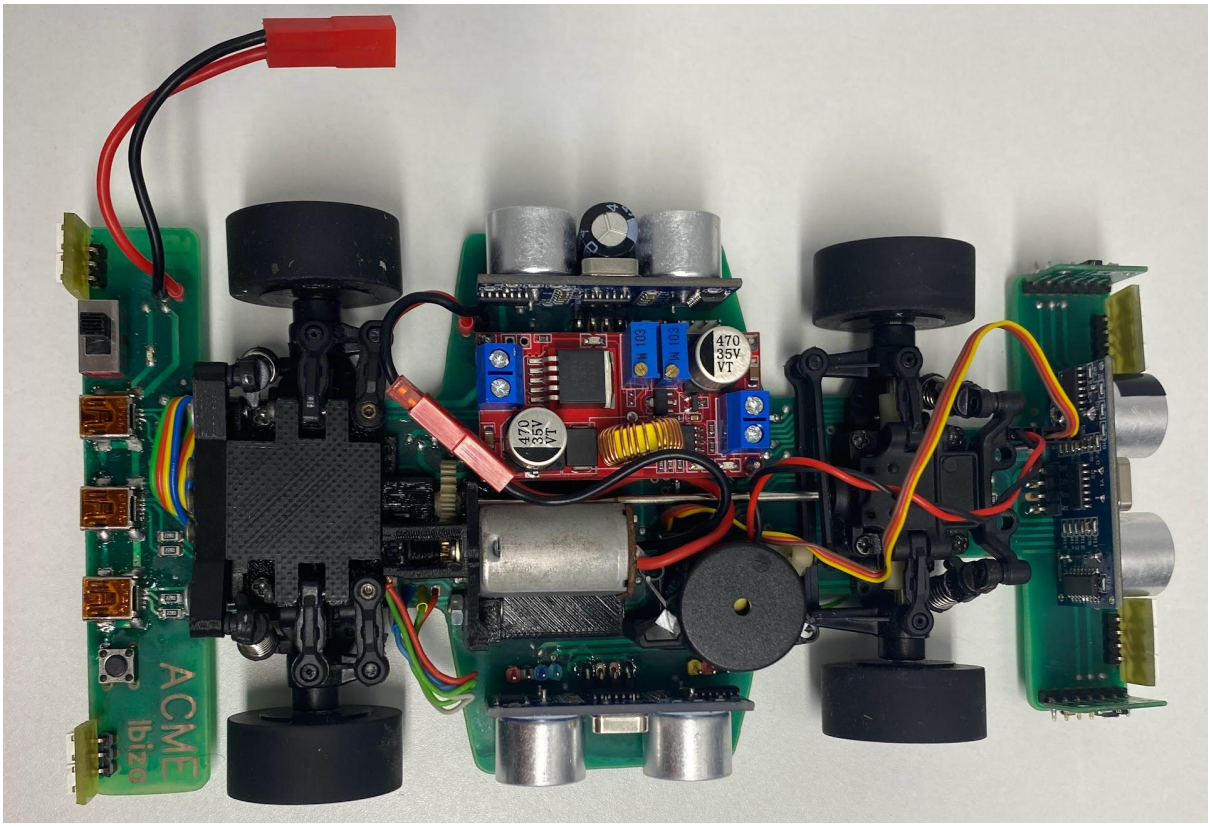

Figure 14. Final vehicle top view.

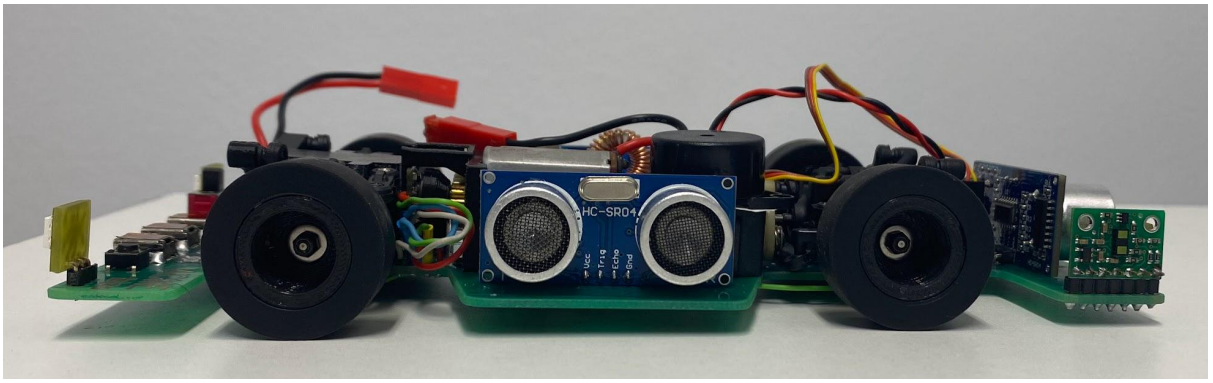

Figure 15. Final vehicle right side view.

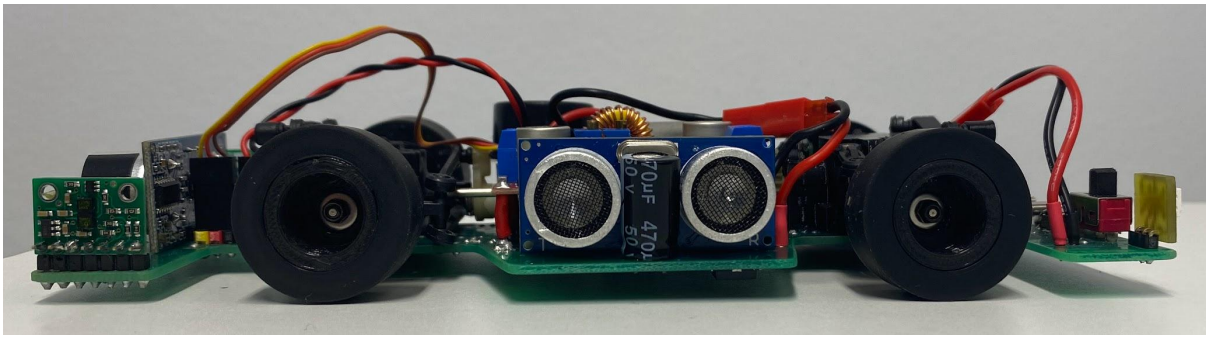

Figure 16. Final vehicle left side view.

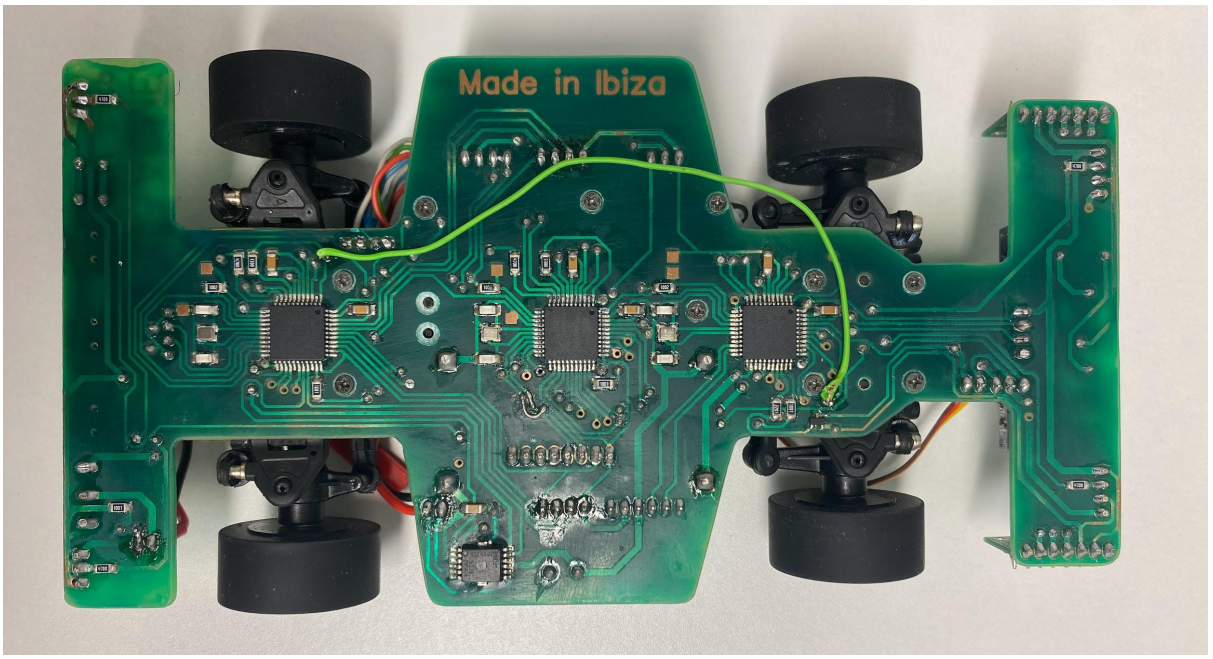

Figure 17. Final vehicle bottom view.

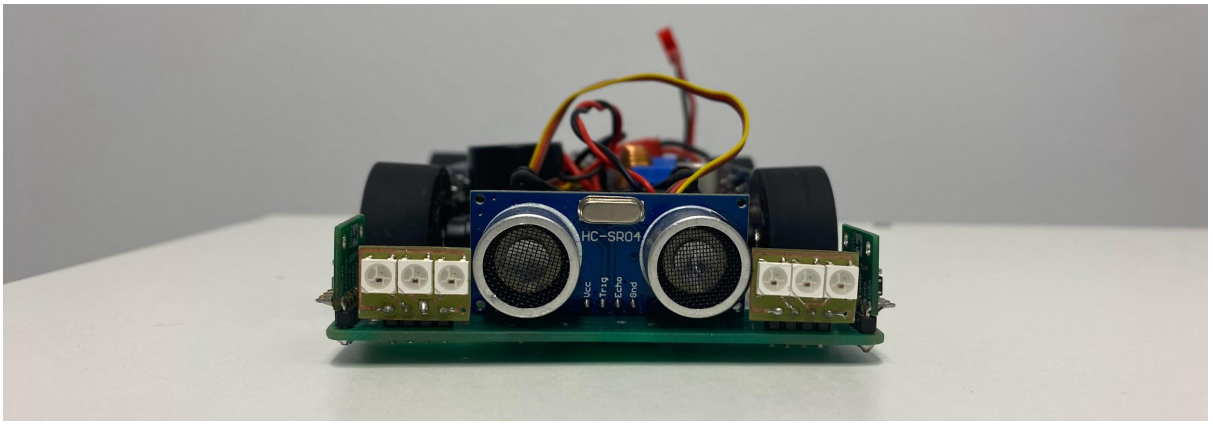

Figure 18. Final vehicle front view.

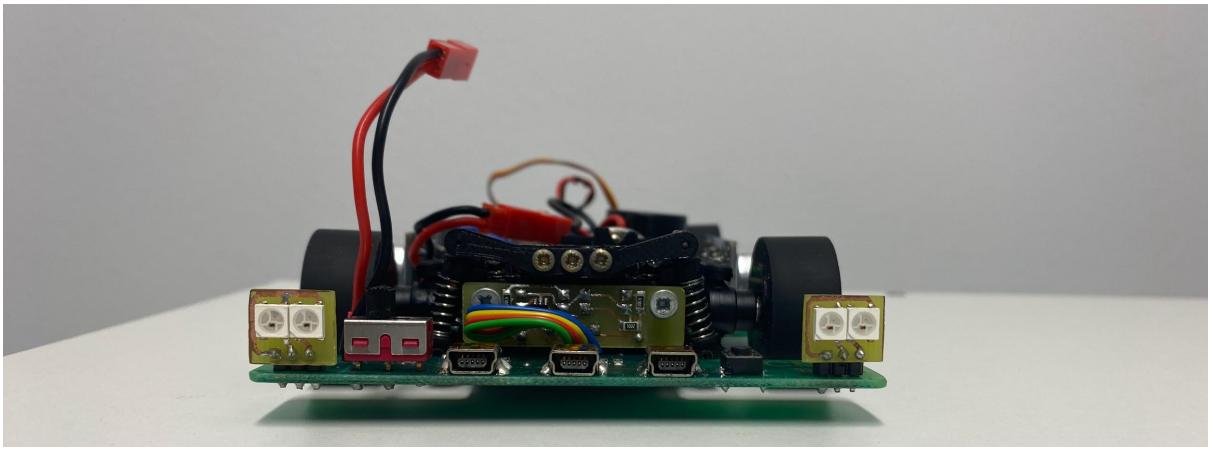

Figure 19. Final vehicle rear view.