Universitat Politècnica de Catalunya

Bachelor's degree in Industrial Electronics and Automatic Control Engineering

Bachelor's Thesis

WIFI REMOTE CONTROLLED RC CAR COMMANDED BY A C# APPLICATION & ESP32 ARDUINO WI-FI MODULE

Author:
Guillem Cornella Barba
Eudald Sangenís Rafart

Supervisor:
Leslie Tekamp
Raúl Benítez Iglesias

University of Colorado Colorado Springs

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

June 22, 2020

Summary

English: In this project, it is expected to build a robot from an RC car that will be controlled through Wi-Fi. The robot will be controlled from a Windows application programmed in the C# language, that will be connected to the ESP32 Wi-Fi module, programmed in Arduino language, to give instructions to the robot. The robot contains an IP camera whereupon its operator can access live to the robot's vision and he can control it from his PC through an Xbox controller.

Key words: Windows application, C#, Arduino, ESP32 Wi-Fi module, IP camera & Xbox controller.

Català: En aquest treball es pretén construir un robot a partir d'un cotxe teledirigit que serà controlat mitjançant Wi-Fi. El robot es controlarà des d'una aplicació Windows programada en llenguatge C#, que es connectarà amb el mòdul Wi-Fi ESP32 programat en llenguatge Arduino, per tal de donar instruccions al robot. El robot conté una càmera IP per la qual el seu operador pot accedir en directe a la visió del robot i el pot controlar des del seu PC mitjançant un controlador Xbox.

Paraules clau: aplicació Windows, C#, Arduino, mòdul Wi-Fi ESP32, càmera IP & controlador Xbox.

Castellano: En este trabajo se pretende construir un robot a partir de un coche teledirigido que será controlado mediante Wi-Fi. El robot se controlará desde una aplicación Windows programada en lenguaje C#, que se conectará con el módulo Wi-Fi ESP32 programado en lenguaje Arduino, para dar instrucciones al robot. El robot contiene una cámara IP con lo cual su operador puede acceder en directo a la visión del robot y lo puede controlar desde su PC mediante un controlador Xbox.

Palabras clave: aplicación Windows, C#, Arduino, módulo Wi-Fi ESP32, cámara IP & controlador Xbox.

Acknowledgements

This project was made possible thanks to the collaboration between Universitat Politècnica de Catalunya (UPC), the University of Colorado Colorado Srings (UCCS) and the funding of the Balsells Foundation.

We would like to thank our advisor Leslie Tekamp from the electrical and computer engineering department for his weekly supervision and for his advice. Last but not least, we would like to thank the UCCS IT department for their readiness at all times.

This project has been challenging and multidisciplinary. Without your help achieving the end results would not have been possible. We thank you all for giving us the opportunity of learning as much as we have learned.

Scope

Project Justification: Among all the proposals provided from Prof. Tekamps, this project was chosen because of our interest in designing and coding an interface that could be able to control a robot through Wi-Fi.

Project Scope: This project consists of creating a remote-controlled robot using a Windows application written in C# that sends data to the robot through Wi-Fi. The microcontroller of the robot is an *ESP32* module and it receives the data from the computer in order to control the robot.

Project Deliverables: At the beginning of the project, a schedule was elaborated in order to know when to deliver the sections that were being done:

MONTH	WEEK	WORK
JANUARY	19 - 25	Create a list of materials and do research
	26 - 1	Buy materials and plan the project
FEBRUARY	2 - 8	Dissemble RC car, understand its functionality and draw the electrical scheme draft
	9 - 15	Create the interface 'PC-camera' using C#
	16 - 22	Create the interface 'PC-robot' using C#
	23 - 29	Test the ESP32 Wi-Fi module and control lights, servos and the motors of the robot
MARCH	1 - 7	Design and 3D print all support Assemble and weld all the components of the robot
	8 - 14	Spring Brake
	15 - 21	Code the behavior of the Xbox controller with C#
	22 - 28	Control the robot with the PC using the ESP32
	29 - 4	Test the robot and resolve errors
APRIL	5 - 11	Write the report
	15 - 21	
	19 - 25	Deliver the report
	26 - 2	-
MAY	3 - 9	-
	10 -16	-

Table 1: Schedule of the project.

Project Success: The project will be determined successfully if the robot runs perfectly using the designed application.

Contents

List of Figures

List of Tables

1 Introduction

Robots are remotely controlled from their beginnings. They have infinity applications in several different camps. In the industry they are usually used to do repetitive tasks and help carrying weights but also they are used for avoiding the direct manipulation of toxic components, in the health field there are robots that help to make surgeries like the appendicitis operation... All these robots have in common two things. The first one is that there is a person who makes the decisions for the robot and order what the robot has to do. The second one is that since the operator has to make a decision, they have to be able to see the vision of the robot. Like Spot [1] a Boston Dynamics robot which is mainly used to carry weight for avoiding causing injuries to workers, which is remotely controlled with a controller where the user can see the vision of the robot in a live feed and it connects to a wireless network to communicate. This robot also has an autonomous mode that records in a video all his activity. However, there are also robots that are autonomous and send afterwards images to the user to make a report. For example, Rovers [2] (NASA's robot which is working in Mars) is able to send images to Earth. On the other hand, rescue robots [3] also have remote-control system but also have an autonomous system because when natural disasters happen, usually the communications also fall down. Even though, it is to important to note that there is always a record of images of the rescue mission.

Figure 1: Spot Boston Dynamics' robot [1].

Figure 2: Rover NASA's robot [2].

The purpose of this report is to explain how it was created a remote-control robot through Wi-Fi from an RC car and be able to reproduce creation in a project class of the University of Colorado Colorado Springs. This will ensure that the students learn about Wi-Fi communications, and the importance of creating robots that can see the environment. The robot was controlled from a Windows application programmed in the C# language. The application sent the datagrams needed to give instructions to the ESP32 Wi-Fi module program in the Arduino language. The robot also contains an IP camera whereupon its operator can access live to the robot's vision and control it from their PC. To maneuver the robot, an Xbox controller is connected to a PC through an USB input. The C# application can read the data from the Xbox controller, process it and send the data packages back to the robot.

2 List of materials

The cells in orange indicate the unused components.
The cells in light red indicate the returned components.

PRODUCT	COST/UNIT	QUANTITY	COST ($)
Arduino Mega	14.99	2	29.98
Arduino pack	34.99	2	69.98
Welding pack	19.99	2	39.98
1:10 Scale Large RC Car	164.97	1	164.97
ESP8266 Wi-Fi module	12.99	2	25.98
Arduino wires	6.98	2	13.96
4 Channel DC 5V Relay Module	7.99	2	15.98
Converter motors	7.49	2	14.98
AMCREST Camera	84.99	2	169.98
L298N Motor Drive	8.89	4	35.56
Arduino Motor Shield	19.99	4	79.96
PCBs	9.99	2	19.98
Servo motor big	19.99	1	19.99
Servo camera support	12.9	1	12.9
XBOX Controller	16.8	1	16.8
Wi-Fi router Linksys AC1000	37.98	1	37.98
12 V power bank	33.99	1	33.99
IP camera	44.99	1	44.99
Cisco camera	299	1	299
TOTAL PAID BY UNI WITHOUT RETURNS			**1146.94**
TOTAL PAYED BY UNI			**677.96**
AC/DC power adapter	12.9	1	12.9
Power bank red	15.99	1	15.99
Power bank black	12.99	1	12.99
Electrical wire	8.38	1	8.38
Switches	6.99	1	6.99
Welding tube	8.79	1	8.79
ESP32 module	14.89	1	14.89
Capacitors pack	10.62	1	10.62
PAID BY US			**91.55**
COST TOTAL AMAZON STUFF			**1238.49**
COST MINUS RETURNED STUFF			**769.51**
COST MINUS NOT USED STUFF			**570.72**
COST OF THE ROBOT			**570.72**

Table 2: List of materials.

3 Hardware

3.1 Motor Driver L298N

In order to control the DC motors [4](LG-DJ01 Laegendary) from the RC car it was cut the connection to the speed controller [5](ZJ07A Laegendary) and the L298N DC motor driver [6] was connected.

To maneuver the robot it is needed to control two variables of the motors, the speed and the direction. This can be achieved by combining two techniques:

- **Pulse Width Modulation (PWM):** this technique allows the user to adjust the average of the voltage by sending a series of ON-OFF pulses. Depending on the duty cycle (relation between the amount of time the signal is ON and OFF in one period) the average voltage supply change.

$$D = \frac{PW}{T} * 100$$

D: duty cycle; PW: pulse active time; T: total time period.

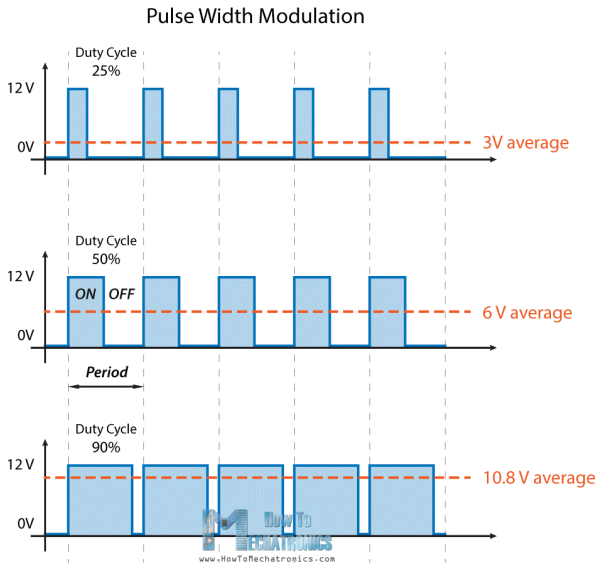


Figure 3: Pulse Width Modulation explanation [7].

- **H-Bridge:** to control the direction (forward & backward) the H-Bridge technique was used. This technique consists of inverse the direction of the current flow through the motor. Seeing the electrical schema (see figure 4), the diagram contains four switches that go odd symmetry, and depending on if one pair of switches are open or closed the flow of the current indicates the direction of the motor. Actually, the switches are transistors MOSFETs but the schema is a simplification.

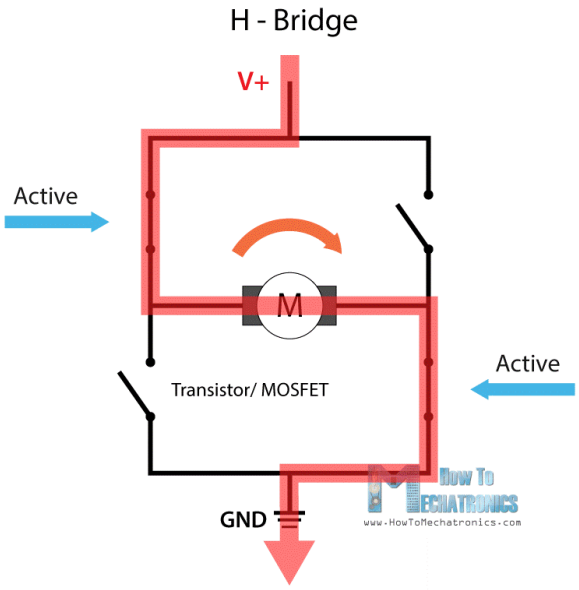


Figure 4: H-Bridge schema using switches to control the rotation direction. [7].

To control the motors the L298N motor driver was used. According to the datasheet of the component this motor driver can control DC motors between 5 and 35 V, with peak current up to 2A. Hence, the motors of the car suit perfectly to this driver because they can receive a maximum input voltage of 7.4 V.

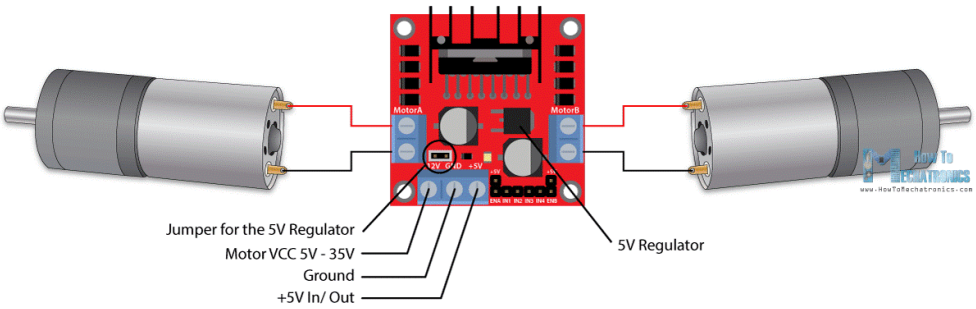


Figure 5: This is the module L298N with two terminal blocks for the motors A and B, and one terminal block for the GND pin, the VCC to power the motor and a 5V pin (input or output).

In figure 5 the connection between DC motors and the module is explained. Focusing in the pinout of the L298N module, this motor driver has a jumper to control the 5 V regulator, the power supply pins, the logic pins, and the output pins.

If the jumper is set, the motors will work at maximum speed. Instead of that, if the jumper is removed, the motors' speed will be controlled by a pulse width modulation through the supplied voltage pin.

There exist two output pins, each one can control only one motor. It must be said that between the input voltage in the input pins and the output pins there exists a 2 V drop. This phenomenon happens for how it is built in this module. That means that if the module is 7 V supplied, then the output is going to be of 5 V.

Finally there are the logic control inputs, as illustrated in figure 6, which can be used to control the speed of the motors through the pulse width modulation (ENA & ENB) and the rotation direction (IN1, IN2, IN3, IN4).

Motor Rotation	**Motor 1 (IN1) Motor 2 (IN2)**	**Motor 1 (IN2) Motor 2 (IN4)**
Stop	0	0
Clockwise	0	1
Anticlockwise	1	0
Stop	1	1

Table 3: How to configured the motor rotation with the pins IN1, IN2, IN3, IN4.

As for motor A, the Input 1 and Input 2 pins are used and for motor B, the inputs 3 and 4. These pins are equivalent to the H-bridge switches. To go forward, the input 1 must be LOW and the input 2 HIGH. To go backward, the input 1 must be HIGH and the input 2 LOW. When both inputs are the same, the motor will stop.

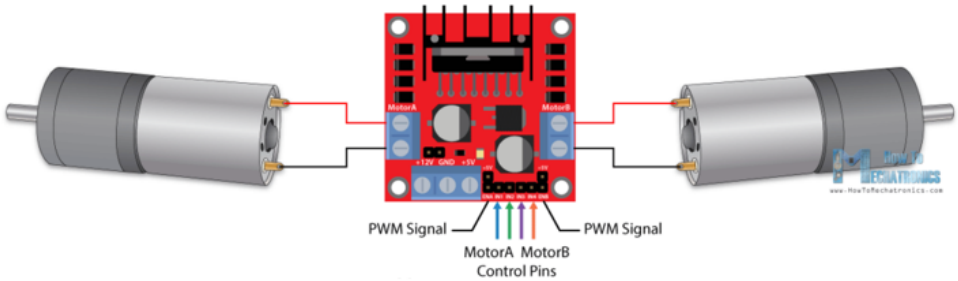


Figure 6: Input pins from left to right: ENA, IN1, IN2, IN3, IN4, ENB.

3.2 Servomotors

The servomotors usually have three wires, two are for the power and the third one is for the signal. The signal is a pulse width modulator (PWM) input. The period usually is for 20 ms (T = 20 ms). And controlling the the duty cycle (DC) can configure the position of the servo. As illustrated in the figure 7, if the duty cycle is 1 millisecond, the servo is positioned at 0 degrees. And if the duty cycle is 2 ms the servo position is the maximum angle possible.

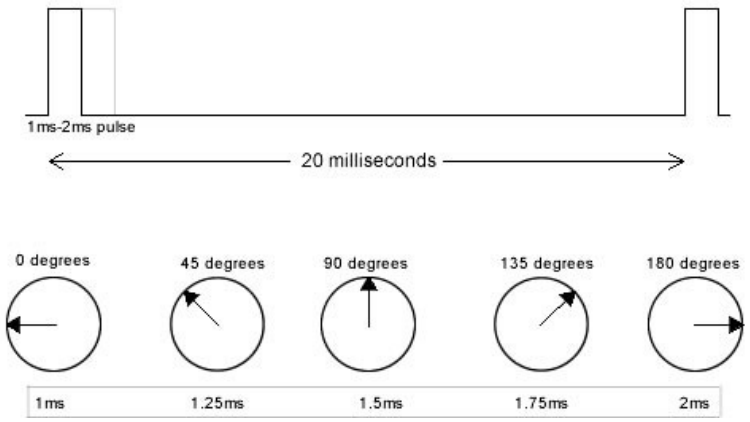


Figure 7: Angle position depends on the length of the pulse [8].

3.2.1 Camera control - MG996R

The *MG996R* servomotor [9] controls the movement of the camera. According to the datasheet, here are some servo characteristics:

Figure 8: MG996R Servomotor.

- **Operating Voltage:** +5V typically
- **Current:** 2.5A (6V)
- **Stall Torque:** 9.4 kg/cm (at 4.8V)
- **Maximum Stall Torque:** 11 kg/cm (6V)
- **Operating speed:** 0.17 s/60°
- **Gear Type:** Metal
- **Rotation:** 0°-180°

3.2.2 Steer control - SG90

The *SG90* servomotor [10] controls the direction of the robot. According to the datasheet, here are some features:

Figure 9: SG90 Servomotor.

- **Stall torque:** 1.8 kg/cm (4.8 V)
- **Gear type:** POM gear set
- **Operating speed:** 0.1 sec/60degree (4.8 V)
- **Operating voltage:** 4.8 V
- **Temperature range:** 0 ℃_55 ℃
- **Dead band width:** 1 us
- **Power Supply:** Through External Adapter

The RC car has a specific place to fit the *LG-ZJ04* servo [11] in to control the steer of the car. Nonetheless, the *"servo.h"* library from Arduino is programmed to control servos with three wires and not with five. So, the main problem was that the majority of the RC car servos have five wires (see figure 12). Finally, it was opted to disassemble the *LG-ZJ04* and try to fit *SG90* inside the casing of the original servo as illustrated in figure 11.

Figure 10: LG-ZJ04 Servomotor.

- **Torque:** 2.2 kg
- **Wires:** 5
- Fits perfectly on LEGEND truck because it is an original accessory of LEGEND RC car

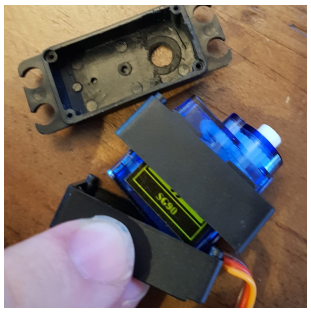


Figure 11: SG90 Servomotor inside the LG-ZJ04.

3.3 Power supply - Battery

Two Li-Po batteries *LG-DJ02* [12] with 1600 mAh, 7.4 V were used. The first one was meant to supply the DC motors and the *ESP32* Wi-Fi module and the second one was used to supply the servos. These batteries are the originals from the Laegendary RC car.

One external battery to supply the servomotors was used since the *ESP32* does not supply the necessary current to power the servomotors. Thus, only one battery is used to feed two servos.

In the first prototype a power bank of 2000 mAh, 5 V was used to supply the servomotors but it did not work because, the power bank does not allow low currents. Therefore, when the servo is stopped it uses 6 mA to maintain the torque, and when the servo is turning, it uses 150 mA. The reason for this bad functioning was because a power bank has an internal battery of 3.7 V and in order to get 5 V from an output there is a step-up converter. Such a circuit draws current even when not loaded with anything and it drains the battery even when not charging anything.

To try to fix the problem of the low current when the servomotor is stopped, to cut the power supply a relay was used. This solution worked well because the consumption was null but also, the response was slow. Also, it was tried to put a resistor in series with the servo to increase the current to 6 mA to 20 mA but when the servo was turning, it did not have sufficient torque to move it. So finally, the *LG-DJ02* Li-Po battery was used.

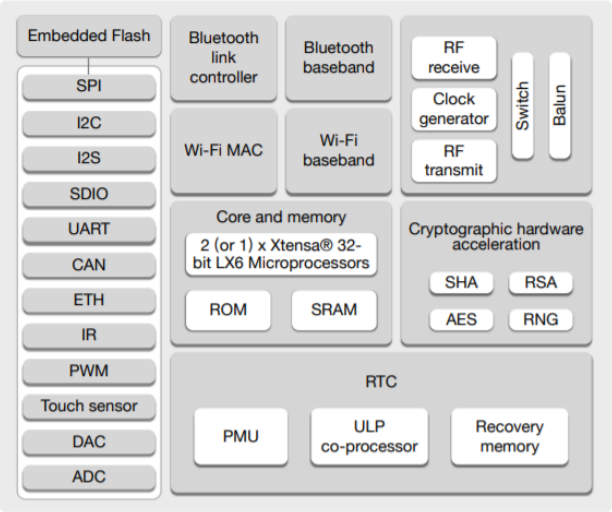
Figure 12: 1600 mAh 7.4V 2S 25C LI-PO Rechargeable Battery [12].

- **Peak performance:** Increase the runtime.
- **Stable performance:** A LiPo will hold a steady voltage for most of your run.
- **More energy density:** LiPo battery have about four times the energy of density of nickel-cadmium or nickel-metal hydride batteries

3.4 ESP32 DEVKITV1 Wi-Fi Module

To communicate the laptop with the robot the *ESP32 DEVKITV1* module [13] was used. The communication via Wi-Fi was established but also, this module has Bluetooth. The ESP32 was chosen because it is easy to implement it with Arduino and have more pins than the *ESP8266* Wi-Fi module.

According to the datasheet, the ESP32 overview specifications are the following ones:



- Dual Core
- RAM of 512 KB
- Frequency of 240 MHz
- Admitted peripherals see figure 13.
- Admit Wi-Fi and Bluetooth communication.
- 30 pins

Figure 13: Block diagram ESP32 [14].

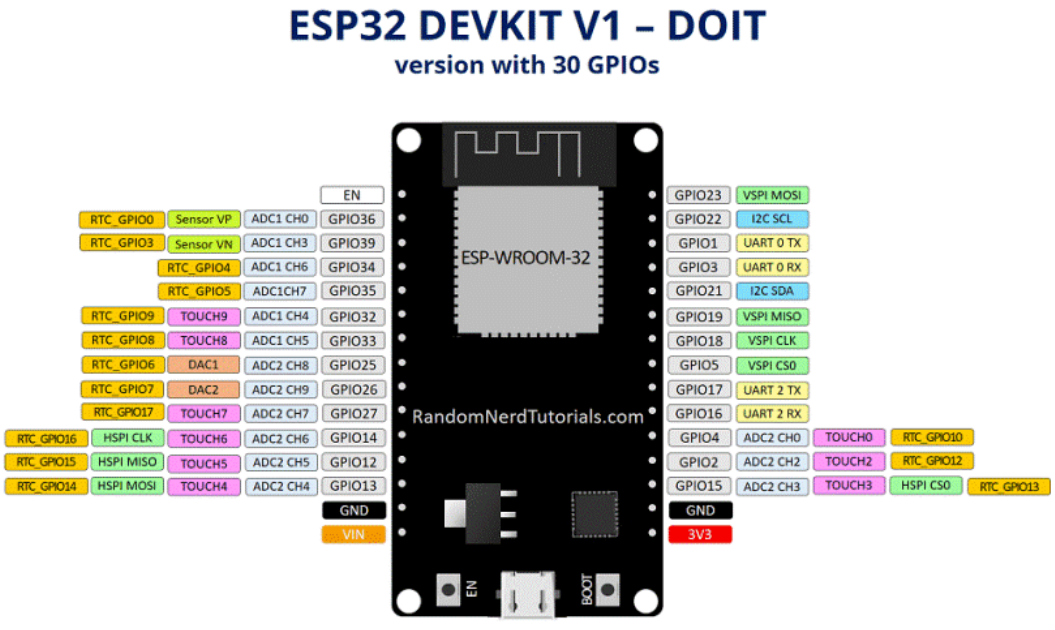


Figure 14: ESP32 DEVKIT V1 - DOIT, version with 30 GPIOs, PINOUT scheme [15].

4 Telecommunications

The purpose of this section is to explain how the module *ESP32* and the camera interact via wireless communication with the computer.

4.1 Network Systems

The wireless communication is one in which the ends of the communication (sender/receiver) are not linked by a physical means of propagation, but rather that, the modulation of electromagnetic waves through space is used. In this sense, the physical devices are only present in the signal transmitters and receivers.

Wi-Fi is a technology that enables the wireless interconnection of electronic devices. The enabled devices (such as personal computers, telephones, televisions ...) can connect to each other or to the internet through a wireless network access point. This communication can be produced in short distances, normally, a Wi-Fi router working on the traditional 2.4 GHz band reach up to 150 feet (46 m) indoors and 300 feet (92 m) outdoors unless there are used repeaters to maximizes the range. But the routers that use the 5.0 GHz band, reach up approximately one-third of these distances. The range is lower in the 5 GHz band because higher frequencies cannot penetrate solid objects, such as walls and floors. However, higher frequencies allow data to be transmitted faster than lower frequencies, so the 5 GHz band allows you to upload and download files faster.

Frequency	**Theoretical Speed**	**Indoor distance**	**Outdoor distance**
2.4 GHz	300 Mpbs	150 feet - 46 m	300 feet - 92 m
5.0 GHz	900 Mbps	50 feet - 15 m	100 feet - 30 m

Table 4: Wi-Fi router properties depending on which band is working on.

So, to communicate the computer to the robot a 2.4 GHz band was used because long distances were needed to run. Thanks to using the 2.4 GHz band, the computer was inside the house and the robot was outside about 100 feet and the communication was fluid.

The router created a LAN[1] to be able to interconnect the applications without needed to connect to the internet as it is illustrated in figure 15. A WAN[2] network is not used because there is no need to cover more space than a campus.

[1]Local Area Network: is a computer network that interconnects computers within a limited area.

[2]Wide Area Network: is a telecommunications network that extends over a large geographical area for the primary purpose of computer networking.

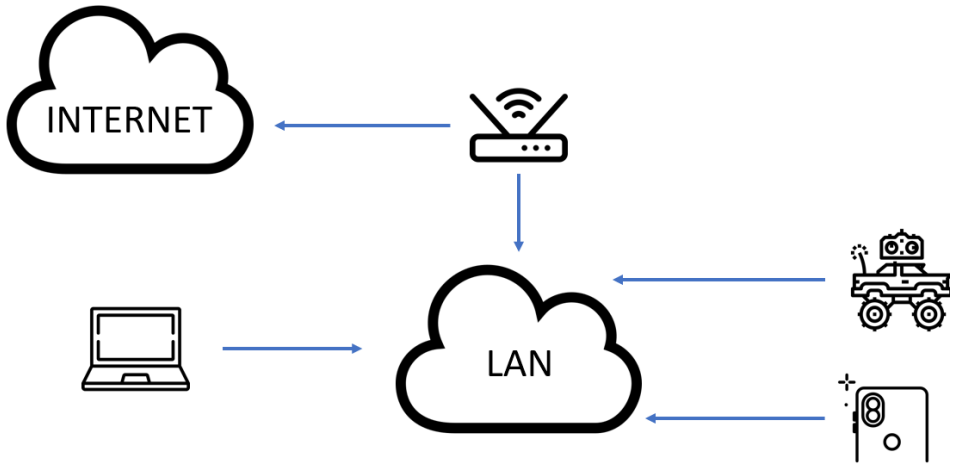


Figure 15: Network structure.

This communication was possible due to how the packet of the data was structured. Those packets were layered as illustrated in figure 16.

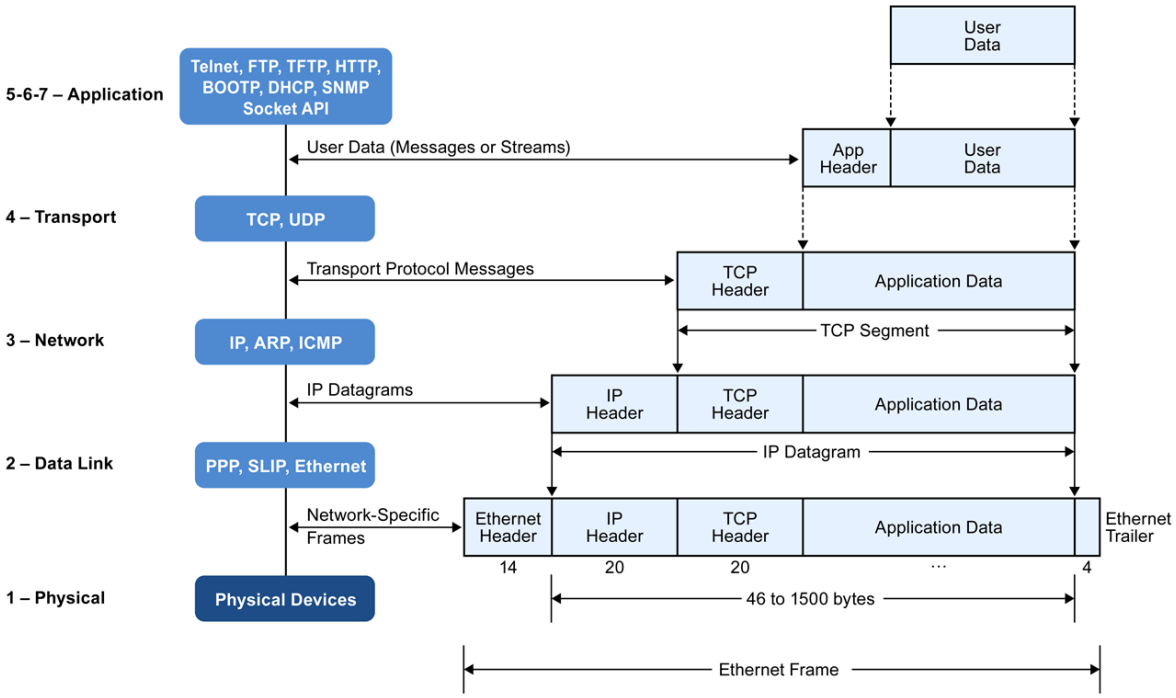


Figure 16: TCP/IP stack [16].

The following example pretends to explain how the stack of the message that the camera sends to the laptop to be able to reproduce the video would be:

The video is the user data.

The camera send the video through a HTTP[3] protocol to distribute the video.

---

1. The application data is the video plus the protocol to distribute it.

The camera has a port (origin port).

The laptop has a port (destination port).

The user data has to be sent it to another device by TCP or UDP.

---

2. The TCP Header is the protocol that indicates how the devices have to communicate, in this case UDP, plus the origin and destined ports.

The camera has an IP (origin IP).

The laptop has an IP (destination IP).

---

3. The IP Header is the origin IP plus the destination IP.

---

4. The Ethernet Header is how the data can be linked, in that case is used a PPP [4] protocol.

---

[3]Hypertext Transfer Protocol: is an application protocol for distributed, collaborative, hypermedia information systems..

[4]Point-to-Point Protocol: is a data link layer communications protocol between two applications directly without any host or any other networking in between.

4.2 Wi-Fi protocols

4.2.1 TCP

The Transmission Control Protocol (TCP) is one of the main protocols of the Internet protocol suite. Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of bytes between applications running on hosts communicating via an IP network. TCP is connection-oriented, and a connection between client and server is established before data can be sent (see figure 17).

The Transmission Control Protocol provides a communication service at an intermediate level between an application program and the Network Protocol (see figure 16). It provides host-to-host connectivity at the transport layer of the Internet model. At the transport layer, TCP handles all handshaking and transmission details and presents an abstraction of the network connection to the application typically through a network socket interface.

A TCP connection is managed by an operating system through a resource that represents the local end-point for communications, the Internet socket. During the lifetime of a TCP connection, the local end-point undergoes a series of state changes as illustrated in figure 17:

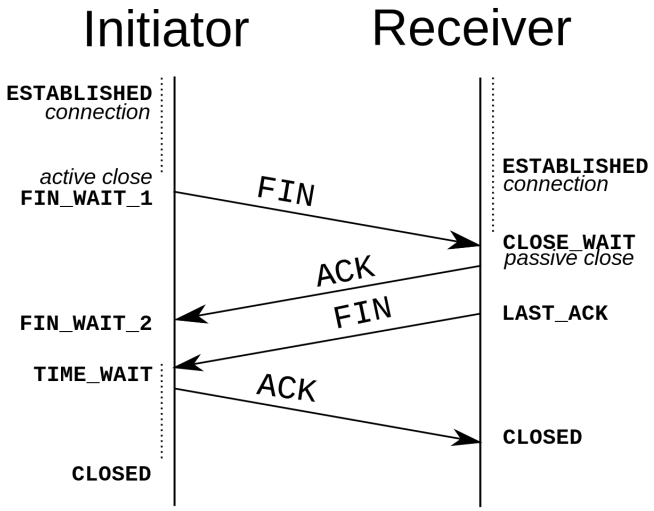


Figure 17: TCP/IP protocol [17].

TCP is used for organizing data in a way that ensures the secure transmission between the server and the client. It guarantees the integrity of data sent over the network, regardless of the amount.

4.2.2 UDP

User Datagram Protocol is mainly used for establishing loss-tolerating and low-latency connections between applications on the internet. It is used as an alternative to TCP protocol.

UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no dialogues between the server and the client (see figure 18), and thus there is no guarantee of delivery, ordering, or duplicate protection. UDP sends data packets (datagrams) to recipients without checking for missed packets. It is used when error correction is not necessary, and speed is desirable. UDP is often used for online games and live broadcasts.

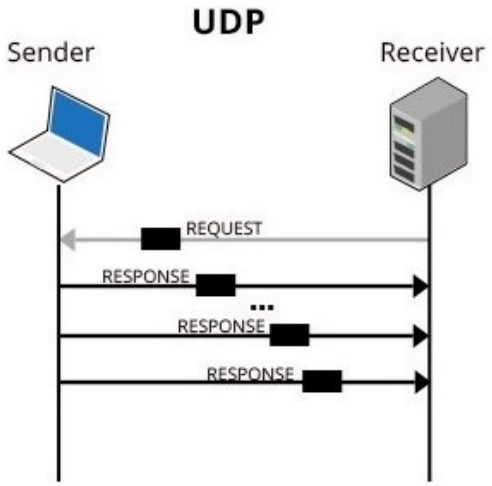


Figure 18: UDP protocol [18].

That properties make the User Datagram Protocol ideal to the project because it is sent two types of data, the camera sends video and the computer sends strings to the *ESP32* module. When the video is sent it is interesting to have a low-latency connection and if a frame of the video is lost is not a big deal. And about the datagrams that the computer sends every 40 ms if a package is lost it makes no difference because the robot has inertia and there is no effect if the interval is upped at 80 ms. So, in that case, it was more important the velocity than the security of receiving the data.

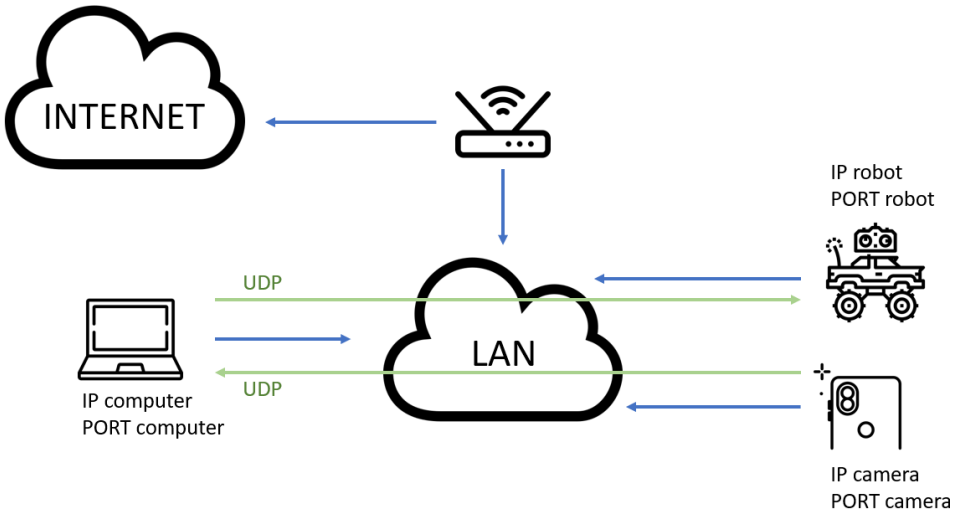


Figure 19: Network structure with protocol communication.

4.3 Network's security

At the university there were three networks, each one with its security:

- UCCS-Guest [19] *"is an open (unsecure) network which does not require any username or password to join - similar to your experience at a place such as a coffee shop. This network does* ***not provide full access****. It offers basic http and https access (for general web browsing), but no access to on-campus resources such as network drives. Advanced functionality is limited (****some ports and applications are blocked****). Should you need access to on-campus resources such as shared network drives, you will need to utilize the UCCS-Wireless network."*

- UCCS-Wireless [19] *"is the full-access secure network intended to be used by all students, faculty, and staff. This network uses* ***WPA2 Enterprise encryption*** *to ensure security. This network provides full access to the Internet. This network requires a username and password to join."*

- Eduroam [19] *"is the secure, world-wide roaming access service developed for the international research and education community. Eduroam allows students, researchers, faculty and staff from participating institutions to obtain Internet connectivity across campuses and when visiting other participating institutions. In other words, if you visit another participating campus, you would connect to the Eduroam wireless network there and enter your UCCS username and password as your credentials in order to gain wireless internet access at that location." This network uses* ***WPA2 Enterprise encryption*** *to ensure security.*

Finally, due to the COVID-19, the project has to be finished with the home's network with a **WPA2 personal encryption** to ensure security. This network provides full

access to the Internet and there are no limits in advance functionalities. This network requires a password to join.

To sum up, these are the differences in the security's types:

- **WPA2 personal:** requires a password
- **WPA2 enterprise:** requires a username an a password
- **Unsecure:** no requisites but with university restrictions.

4.4 Device connection

	UCCS-Guest Unsecure	UCCS-Wireless WPA2 enter.	Eduroam WPA2 enter.	Home Network WPA2 per.
Amcrest	No connect	No connect	No connect	Connect
Cisco	No connect	Connect	No connect	Connect
Cell Phone	Connect	Connect	Connect	Connect
ESP32	Connect	No connect	No connect	Connect

Table 5: The existing types of Wi-Fi and the ability to connect of the devices.

After evaluating the options, the cell phone's camera was the best option to connect to the UCCS-Guest and therefore it was chosen but it was not able to visualise the images on the application. Seeing that, the IT department was asked and said that to send video with this network was not possible due to the university policies. The university does not want the students to install cameras on campus so, the ports of the network were blocked.

Therefore, the IT department proposed to create a network only for this project with WPA2 personal security but owing to the emergency sanitary of COVID-19 the university had to be closed. Hence, the project was required to be finished at home. Finally, due to the AMCREST and Cisco camera had been returned it was chosen to use the cell phone camera to lower the cost of the project.

5 Application Code

The Visual Studio v.2019 (therefore onwards VS) was decided to use as the code editor of the application because it allows the developer to code easily computer programs, as well as websites, web apps, web services, and mobile apps. VS uses Microsoft Software development platforms such as Windows API and Windows Forms.

VS is the code editor to make the application for Windows, creating a *WPF (.NET Framework)* project. WPF is useful because it allows the programmer to separate the design from the logic. It employs XAML language for the design and for the logic it uses principally C# code but also it can be combined with MySQL if you are working with databases or other languages. So, following the rules of the WPF projects, the project is structured as illustrated in figure 20:

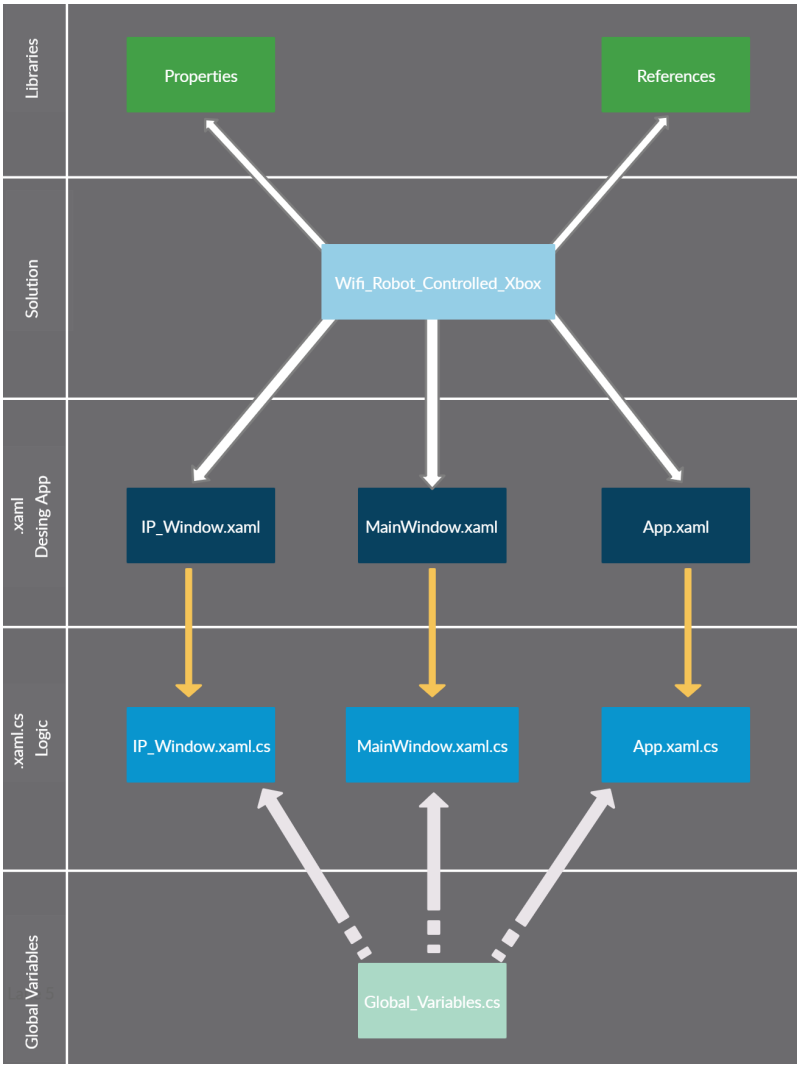


Figure 20: Structure code.

5.1 Libraries

In order to be able to program the application, some functions from libraries were used. Mainly there existed two options. The first one was to download the packages that were needed from NuGet[5]. The other one was to just call the libraries that were already downloaded when they have been installed the VS.

5.1.1 NuGet libraries

To develop the GUI[6] some Nu packages from NuGet were needed to be installed. Usually, when these packages are automatically installed, they are added to references. However, if they have already been installed and a new project is created, it is needed to add the *.dll* document to the references. To build the application some packages have been needed to be installed:

- **MaterialDesingColors** – to be able to have templates for design the interface.
- **VisioForge** – to be able to reproduce the video in the API.
- **SharpDx** – to be able to install the Nu package SharpDX.Input.
- **SharpDX.Input** – to be able to read inputs from the COM (Xbox Controller).
- **OpenJiWare** – to be able to interpret the values from the Xbox Controller.

After that, it is required to include them in your program. In the case of MaterialDesingColors because it is related to the design of the GUI, it has to be called through the App.xaml as illustrated in the following code fragment:

```
<Application.Resources>
    <ResourceDictionary>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary Source="pack://application
            :,,,/MaterialDesignThemes.Wpf;component/Themes/
            MaterialDesignTheme.Light.xaml" />
            <ResourceDictionary Source="pack://application
            :,,,/MaterialDesignThemes.Wpf;component/Themes/
            MaterialDesignTheme.Defaults.xaml" />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

Listing 1: Import MaterialDesingColors NuGet package in the App.xaml.

[5]NuGet is the package manager for .NET.
[6]GUI: Graphic User Interface

Nevertheless, the other packages are easier to be included in the libraries of the references of the documents. In the MainWindow.xaml, the following ones are used:

```
using VisioForge.Types.OutputFormat;
using OpenJigWare;
```

Listing 2: Import VisioForge & OpenJigWare NuGet package in the MainWindow.xaml.

Finally, there is no need to call the SharpDX.Input, but it has to be installed if it has to be avoided a break down of the program. The main reason behind it is that the program is not going to be able to read the inputs from the Xbox controller. This error happens because inside the OpenJiWare library there are references to commands from SharpDX.Input.

5.1.2 System libraries

The System libraries contain fundamental classes and base classes that define commonly used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions. In the program only the following ones are needed:

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Text;
using System.Timers;
using System.Net.Sockets;
```

Listing 3: System libraries that is needed to import in the project.

- **System.Windows.Controls:**
  Provides classes to create elements that enable a user to interact with an application.
- **System.Text:**
  It contains classes that represent ASCII and Unicode character encoding to treat the data.
- **System.Timers:**
  The timer is useful to call the function to read the values from the Xbox Controller every 40 ms.
- **System.Net.Sockets:**
  The Socket library is necessary to establish the UDP connection between the robot and the application.

5.2 Camera characteristics & code

First of all, to determine which camera had to be used to display the vision of the remote-controlled car the university's IT department was asked. They required some specifications that the camera had to accomplish.

- Connect via IP
- Be able to connect to 2.4GHz
- Be able to connect to Wi-Fi with a security WPA2

With those details it was decided to buy the AMCREST IP2M-841B(see figure 21). This camera allowed the control not only of the video but also the pan, the tilt and the zoom.

Figure 21: Camera AMCREST IP2M-841B

When it was tried at home it worked perfectly but, when it was proved at the university's wireless network it did not connect. Consequently, the IT department updated the given security details specifying that the security network is *WPA2 enterprise*.

Based on the new specifications, some cameras were researched and found that the best camera that complied with the details was the Video Surveillance IP Cameras from Cisco (see figure 22). Following the discovery the model 2500 of the camera was bought, but when it arrived some parts of it were missing. In order to try to search for a solution, it was decided to go to the Cisco department in Colorado Springs, but they were not able to be of any assistance. So, it was agreed to return the camera to Cisco and try to find another solution.

Figure 22: Cisco Surveillance IP Camera 2500

Finally, it was opted to use the camera of a mobile phone. So, Reading this article titled *Using an Android phone as a webcam* from GitHub [20] suggested that downloading the *IP Webcam* App from the Play Store and configuring some parameters, the camera from a phone would be able to be transformed to an IP camera. It was a perfect solution since the mobile phone connects to the UCCS-Guest, UCCS-Wireless or Eduroam networks easily.

The code starts with the design code implemented in the *MainWindow.xaml*. Firstly, it is needed to call the library of the *VisioForge* mentioned earlier as is written in the following fragment of the program.

```
xmlns:WPF="clr-namespace:VisioForge.Controls.UI.WPF;
           assembly=VisioForge.Controls.UI"
```

Listing 4: XAML implementation of the VisioForge library to create the view.

After the library is implemented it is possible to create the view of the camera in the GUI.

```
<WPF:VideoCapture Name="videoCapture1"
                  HorizontalAlignment="Left"
                  VerticalAlignment="Top"
                  Height="336" Width="450"
                  Grid.Column="1" Grid.Row="1">
</WPF:VideoCapture>
```

Listing 5: XAML implementation to create the view interface.

Then there is the behind code of the design in the *MainWindow.xaml.cs* document. Four event functions are created, and they raise the event when the buttons are pressed. The following fragment of code shows the headers of these event functions:

```
private void Connect(object sender, RoutedEventArgs e);
private void Disconnect(object sender, RoutedEventArgs e);
private void Snapshot(object sender, RoutedEventArgs e);
private void Record(object sender, RoutedEventArgs e);
```

Listing 6: Headers of the functions to control the camera.

The first one is used to establish a connection between the app and the phone and display the image. The communication is made through a URL[7] given by the *IP Webcam App*. Also, the function throws error messages in an attempt to not breakdown the code. In the following fragment of the *"Connect"* function shows how connect the camera.

[7]Uniform Recurs Location colloquially termed a web address is a reference to a web resource that specifies its location on a computer network

```
///initialize the image
videoCapture1.IP_Camera_Source = new
VisioForge.Types.Sources.IPCameraSourceSettings()
{
    ///Especific URL to connect to S8
    URL = "http:/" + Global_Variables.ip_cam + ":"
    + Global_Variables.port_cam.ToString() + "/video",
    ///Type HTTP: we obtein the image through
    ///and URL, LowLatency: we want the image
    ///in real time, but we lost resolution.
    Type = VisioForge.Types.VFIPSource.
           HTTP_MJPEG_LowLatency
};
///initialize the audio
videoCapture1.Audio_PlayAudio = false;
videoCapture1.Audio_RecordAudio = false;
///Mode IP cam
videoCapture1.Mode = VisioForge.Types.
VFVideoCaptureMode.IPPreview;
connect_cam = true;
///Method to start connection
videoCapture1.Start();
```

Listing 7: Fragment of the *"Connect function"*.

The second one is used to cut the connection and disable the video when the *Disconnect Button* is pressed. So, when the method *".Stop()"* it is used the connection stops.

```
///Method to stop the connectivity
videoCapture1.Stop();
```

Listing 8: Fragment of the *"Disconnect function"*.

The third one is used to take a picture of the view and save it to the indicated folder of your computer.

```
///Number of snapshot button clicks
count_btn_snap++;
///Save the snapshot to mypictures folder
videoCapture1.Frame_Save(Environment.
    GetFolderPath(Environment.SpecialFolder.
    MyPictures) + $"\\frame_{count_btn_snap}.jpg",
    VisioForge.Types.VFImageFormat.JPEG, 85);
```

Listing 9: Fragment of the *"Snapshot function"*.

Finally, the fourth one is used to record a video of the robot's view and save it to the indicated folder. This function is similar to the *"Connect"* function but it has to be added the video format, the type of video and the way how to save it. Also, this function has to have into account if the connection is already set up to close it and restart it.

```
///number of record button clicks
count_btn_recd++;
///if the streaming is enable it close
if (videoCapture1.IsEnabled) videoCapture1.Stop();
///Create a new videoCapture1 object like the connect func.
...
///Audio Settings like the connect function
...
///save the video to myvideos folder
videoCapture1.Output_Filename =
Environment.GetFolderPath(Environment.SpecialFolder
.MyVideos) + $"\\vid_{count_btn_recd}.mp4";
///Output video format
videoCapture1.Output_Format = new VFWMVOutput();
///Type of video
videoCapture1.Mode = VisioForge.Types.
                     VFVideoCaptureMode.IPCapture;
///Method to start connection
videoCapture1.Start();
```

Listing 10: Fragment of the *"Record function"*.

5.3 Controller code

To be able to control the robot two applications were designed. The first one was a prototype because a platform was needed to try to send data to the robot. In this prototype the robot was controlled through the laptop's keyboard. The second one was the final application controlled with the Xbox controller.

5.3.1 Keyboard

Firstly, an application to be able to send data to the *ESP32* module and see how it responded was designed. The main problem that was faced was the maneuverability of the robot. To pilot the car the velocity and the steer are needed to be controlled. Hence, a packet of data (datagram) sent via UDP protocol to the *ESP32* every 40 ms. This packages contained the information to say to move forward *"x"* distance when the event of the key *"w"* was *key_up* and the velocity. So, to control the steer four buttons were created, and to manipulate the velocity a slicer was designed As illustrated in figure 23.

This type of control created two problems. The first one was that the robot did not change the direction unless the *"x"* distance had finished and if the robot wanted to continue to keep forward, the *"w"* key was needed to press again. The second one occurred when the velocity was tried to be set up because it was not very intuitive and dynamic.

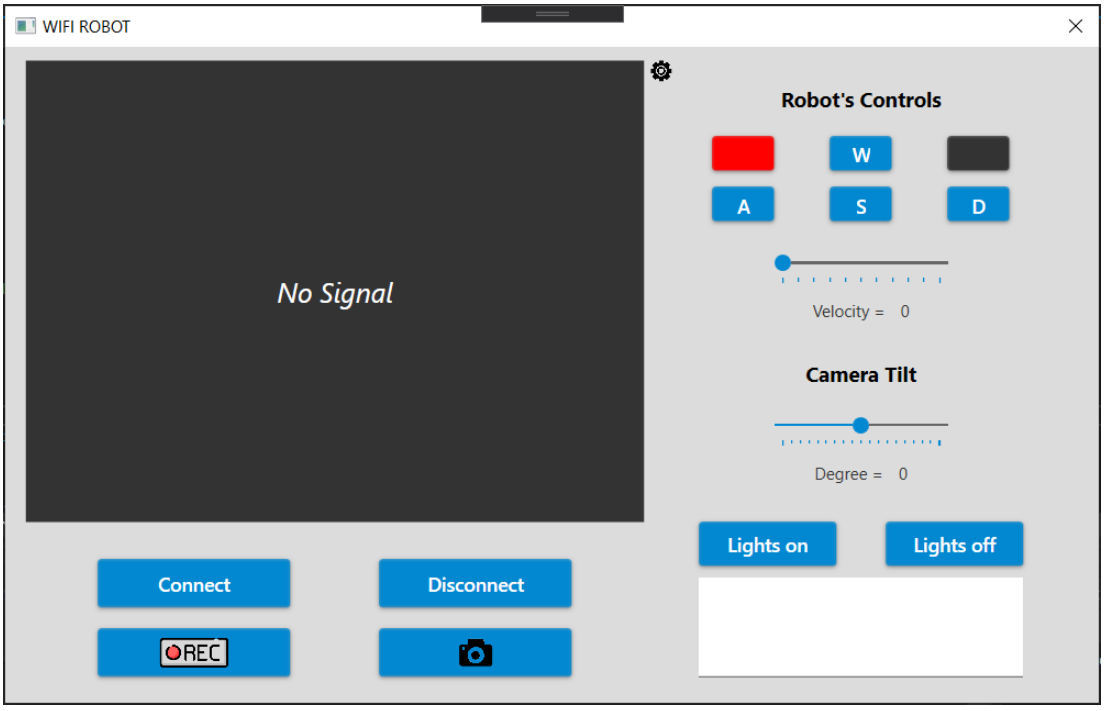


Figure 23: Keyboard Controller application interface.

The following fragment of code illustrates how to declare a private function to raise buttons events when a keyboard key is pressed.

```
 /// Keyboard control
private void key_event(object sender, KeyEventArgs e)
{
    switch (e.Key)
    {
        case Key.A:
            BtnA.RaiseEvent
            (new RoutedEventArgs(Button.ClickEvent));
            break;
        /// Changing the Key.A, the keyword key is changed
        ...
    }
}
```

Listing 11: Fragment of the function to raise buttons events when a keyboard key is press.

The next fragment of code is an example of how the application is able to send the datagrams via UDP protocol to the *ESP32* when button *A* is pressed.

```
///Text to send to the arduino
Global_Variables.cmd = "A";
///Connect to the ip & port of the ESP32
udpSender.Connect(Global_Variables.ip_rob,
                  Global_Variables.port_rob);
///Transform to codi ASCII the "A"
Byte[] sendBytes =
Encoding.ASCII.GetBytes(Global_Variables.cmd);
///send "A" in codi ASCII
udpSender.Send(sendBytes, sendBytes.Length);
```

Listing 12: Fragment of the *"A function"*

From this example is easy to extrapolate how functions should be programmed to go forward, backward, and rightward.

To connect to the IP and the port of the ESP32 device, the *IP_Window* (see in figure 43) was designed. This interface asks for the camera's and the robot's IP number and the camera's and robot's port number to be set up. Those values had to be entered manually and saved clicking to the *"Apply"* button.

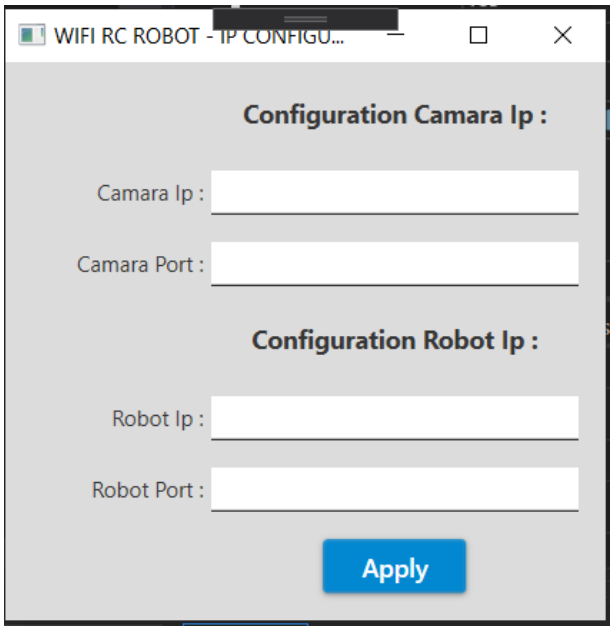


Figure 24: IP Window interface.

5.3.2 Xbox Controller

Once the robot had been driven, improve the maneuverability of the car buying an Xbox controller was decided. So, another interface (see figure 42) was decided to be built but maintaining the *IP_Window* interface (see figure 43). The Xbox commander is constituted by two joysticks with two-axis each one (jx0, jy0, jx1, jy1) and 14 buttons (four arrows, back, start, X, A, B, Y, RB, LB, RT, and LT). It is only used ten commands over eighteen possible to control the robot:

- **X** - Connect camera
- **Y** - Disconnect camera
- **A** - Available start driving
- **RB** - Film video
- **LB** - Take snapshot
- **jx0** - Camera tilt
- **jx1** - Steer
- **jy1** - Speed
- **Left arrow** - Turn off light
- **Down arrow** - Turn on light

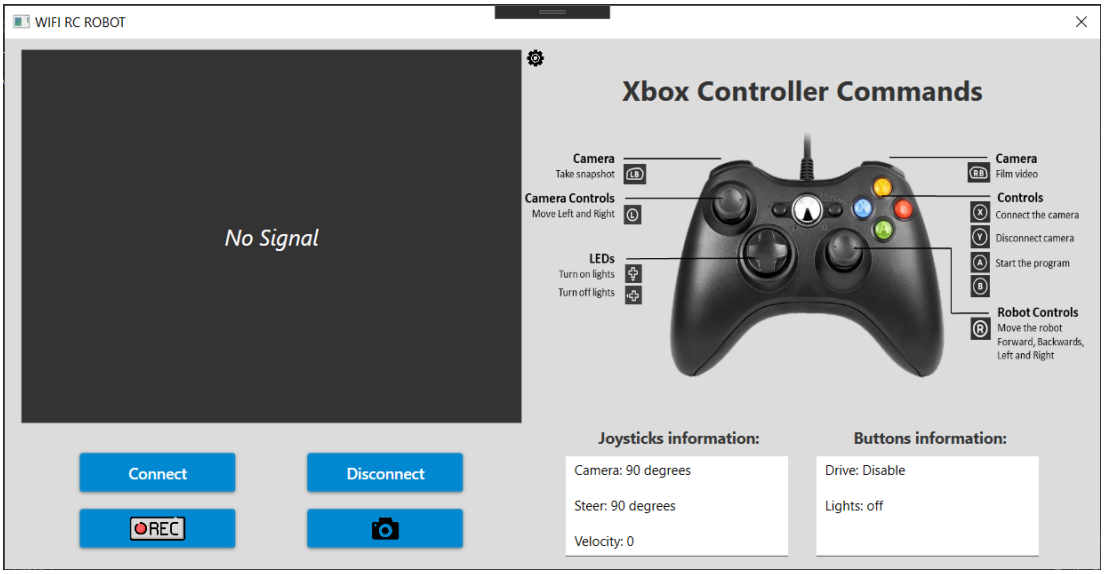


Figure 25: Xbox Controller application interface.

In order to read the joysticks values, a timer object that raises his event every 40 ms and set it up when the *main window* interface is initialized was required.

```
///create a new timer object
private static Timer timer;

public MainWindow()
{
    InitializeComponent();
    ///Set timer up
    timer = new Timer();
    ///Interval 40ms
    timer.Interval = 40;
    ///Call the function when the event raise.
    timer.Elapsed += new
    ElapsedEventHandler(this.timer_Tick);
    ///Enable the timer
    timer.Enabled = true;
}
```

Listing 13: Create a timer object and set it up.

Every 40 ms was called the *timer_Tick* function. This one calls the *m_CJoy.Update* function to update the data from the joystick and from the *Joystick_Check_Data* function to read the data, processed and send it to the *ESP32* module.

```
///Joystick Declaration
private Ojw.CJoystick m_CJoy = new
Ojw.CJoystick(Ojw.CJoystick._ID_0);

public void timer_Tick(object sender, EventArgs e)
{
    // update joystick information
    m_CJoy.Update();
    // Joystick Data Check
    Joystick_Check_Data();
}
```

Listing 14: $\text{timer}_{T}ick function$.

As was said before, to read the data, processed, and send it to the *ESP32* module the *Joystick_Check_Data* function was used. So, the function is divided by regions (arrows, buttons, RB and LB buttons, joystick values, and send and print data to the text boxes of the interface).

The following fragment of code is just a few lines of the function *Joystick_Check_Data*, just to give an idea of how to program this function. This example shows how to call the function *Connect* if the X button from the Xbox controller is pressed, how to obtain the joysticks information, and finally, how to send the information through the UDP protocol.

```
///BUTTON X
if (m_CJoy.IsDown_Event(Ojw.CJoystick.PadKey.Button3)
== true && XBOX_X == false)
{
    this.Dispatcher.Invoke(() =>
    {
        BtnConnect.RaiseEvent(new
        RoutedEventArgs(Button.ClickEvent));
    });
    XBOX_X = true;
}

//DATA FROM JOYSTICK
///camera servo
jx0 = Math.Round(m_CJoy.dX0, 1).ToString();
///steer  servo
jx1 = Math.Round(m_CJoy.dX1, 1).ToString();
///velocity
jy1 = (1-Math.Round(m_CJoy.dY1, 1)).ToString();
```

```
//SEND DATA TO ESP32
///information to send
var joystick_values = "jx0" + jx0 + "jx1" + jx1 + "jy1" +
jy1 + "LED" + leds;
///Establish the connection with the ip & port
udpClient.Connect(Global_Variables.ip_rob,
Global_Variables.port_rob);
///Transform the information to ASCII
Byte[] sendBytes_jx0 = Encoding.ASCII.
GetBytes(joystick_values);
///Send the information
udpClient.Send(sendBytes_jx0, sendBytes_jx0.Length);
```

Listing 15: Joystick_Check_Data function.

Finally, to see the full code implementation is possible to access in the following GitHub [21].

6 Arduino Code

The purpose of this section is to explain how the Arduino code works step by step. In other words, how the ESP32 receives the data from the laptop, how it processed and which is the response.

Firstly, the ESP32 board was installed in the Arduino program.

1. In your Arduino IDE, go to File – Preferences.
2. Enter https://dl.espressif.com/dl/package_esp32_index.json into the "Additional Board Manager URLs" field. Then, click the "OK" button.
3. Open the Boards Manager. Go to Tools ¿ Board ¿ Boards Manager...
4. Search for ESP32 and press install button for the "ESP32 by Espressif Systems".
5. It should be installed after a few seconds.

Secondly, the ESP32 libraries were installed and declared.

```
#include <WiFiUdp.h>
#include <WiFi.h>
#include <ESP32Servo.h>
#include <MapFloat.h>
```

Listing 16: Libraries necessary for Arduino code.

- **WiFiUdp.h:** This library is used to create the instances necessary to send and receive UDP packets.
- **WiFi.h:** This library does not support the WPA2-Enterprise security, only allow to connect to the networks with WPA2-Personal security (SSID & password).
- **ESP32Servo.h:** This library permits the ESP32 board to control the servomotors using the ESP32 PWM timers programming with Arduino.
- **MapFloat.h:** This library contains the mapFloat function which re-maps a floating-point number from one range to another.

The next step after declaring the libraries was to define the variables that were used during the program. In the following fragment of code is possible to see how the name of the router and the password can be declared.

```
const char* ssid = "name_of_router ";
const char* password = "password_goes_here";
```

Listing 17: Defining the router variables.

6.1 Setup

The connection between the WiFi module and the network needed to be completed. The begin method was called on the WiFi object, passing as arguments the SSID and the password variables. This line started the connection to the network:

```
WiFi.begin(ssid, password);
```

Listing 18: Start to try to connect to the network specified.

After beginning the connection it was necessary to wait until the ESP32 was connected. To do so, a while loop was created to print in the serial window a list of points every 100 ms to show that the program is waiting to connect to the network.

```
while (WiFi.status() != WL_CONNECTED)
{
    Serial.print(".");
    delay(100);
}
```

Listing 19: Waiting the connection to be established.

Once the connection was done, the serial monitor printed the IP of the module and the port. These values in the *IP Window* of the C# application were uploaded to establish a connection between the PC with the *ESP32*.

```
Serial.println(WiFi.localIP());
udp.begin(udpPort);
Serial.println(udpPort);
```

Listing 20: Print the IP and the Port in wich the ESP32 module is connect.

Now the program was ready to run its main loop to control the outputs of the robot (lights, servo motors and DC motors).

6.2 Loop

The C# application every 40 ms was sending the data received by the Xbox controller to the Arduino module through an UDP protocol. This protocol was basically sending a packet of information in ASCII format using the IP and port of the module. This packet of data is a String type of 22 characters. Every character uses 1 byte (8 bits), then the packet in total uses 8*22 = 176 bits or 22 bytes from a maximum capacity of 255 bits.

```
int packetSize = udp.parsePacket();
if (packetSize)
{
    IPAddress remoteIp = udp.remoteIP();
    int len = udp.read(packetBuffer, 255);
    if (len > 0) packetBuffer[len] = 0;
    String udp_packet_sent = packetBuffer;
    }
```

Listing 21: Transform the ASCII packet information to string.

This String was named *"udp_packet_sent"* and it contained 4 sub-packages of information which needed to be separated in order to obtain the information needed for each output. For instance, as for the servo controls, a substring was created in order to obtain information regarding the tilt value expressed as *"jx0—"*. The program then converted these Strings into 'int' variables.

```
String val_tilt_camera = udp_packet_sent.substring(0,6);
val_tilt_camera = val_tilt_camera.substring(3,6);
float val_tilt_camera_f = val_tilt_camera.toFloat
val_tilt_camera_f = val_tilt_camera_f - 0.5;
float servo_cam_angle =
mapFloat(val_tilt_camera_f, -0.5, 0.5, 170.0, 10.0);
int servo_cam_angle_int = (int)servo_cam_angle;
```

Listing 22: Processing the tilt data from the string *udp_packedt_sent*.

The same structure was used for the Servo that is controlling the steer.

To light on the lights of the robot a substring was also used in order to obtain just the information needed. Then, the program analyzed the String and compared it with the 3 different options available. If it had received "LED1" for example, it would create a Boolean variable called 'led' and then the light could turn on.

```
String light = udp_packet_sent.substring(18,22);
if (light=="LED1")
{
    led = true;
    digitalWrite(LED,HIGH);
}
else if (light=="LED2")
{
    led = false;
    digitalWrite(LED, LOW);
}
else if (light=="LED0"){
    digitalWrite(LED,LOW);
    if (led==true) digitalWrite(LED,HIGH);
    else digitalWrite(LED,LOW);
}
```

Listing 23: Processing the LEDs data from the string *udp_packedt_sent*.

In order to go forwards and backwards, the first step was to convert the string received by the computer into an 'int' variable, the same procedure was used for the servos. Then, the program distinguished from positive values (forward) and negative values (backwards), ranged between -0.5 and 0.5. Finally, the code was mapped to reconvert the float value into the velocity value, which was saved as an 'int'.

```
// to go FORWARD
else if (val_vel_f > 0.1)
{
  // Set Motor A forward:
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  // Set Motor B forward:
  digitalWrite(in3, LOW);
  digitalWrite(in4, HIGH);

  float motorSpeedA_f =
  mapFloat(val_vel_f, 0.1, 0.5, 400.0, 1000.0);
  float motorSpeedB_f =
  mapFloat(val_vel_f, 0.1, 0.5, 400.0, 1000.0);
  motorSpeedA = (int)motorSpeedA_f;
  motorSpeedB = (int)motorSpeedB_f;
}
```

Listing 24: Processing the velocity for each motor.

The last lines of the code enabled both motors which run at a specific speed.

```
// Send PWM signal to motor A
analogWrite(enA, motorSpeedA);
// Send PWM signal to motor B
analogWrite(enB, motorSpeedB);
```

Listing 25: Enable motors at the correspondence velocity.

7 Building the robot

In this chapter, a detailed explanation of the construction process of the robot is going to take place. Step by step, the construction process consisted on:

1. Research and material acquisition.
2. Disassemble the RC car.
3. Design of a 3D support.
4. Initial connections in the breadboard and first test of the code.
5. Weld the components to the PCB circuit board.
6. Final base design and 3D supports.
7. Assembling.
8. Electrical diagram.
9. Final tests.

7.1 Research and material acquisition

A first brainstorming and research process through the internet was done in order to find the best materials possible with the project budget. Using the economical resources provided by our undergraduate instructor, the following materials had collected:

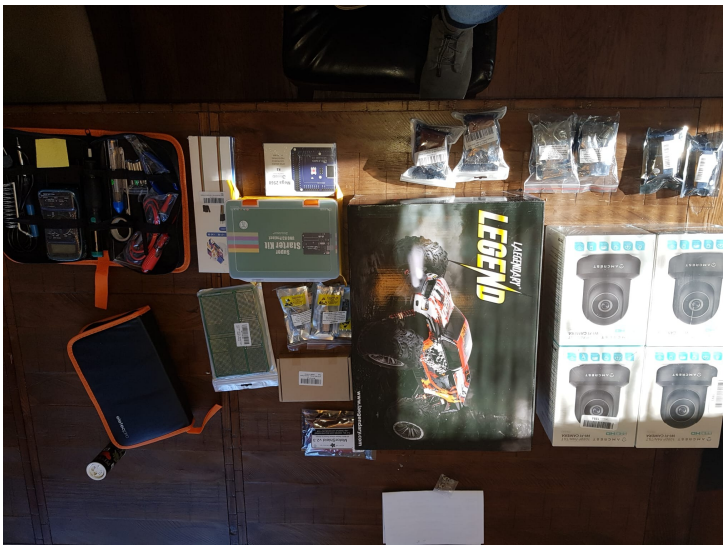


Figure 26: Acquisition of the materials.

For any further information of the materials that were bought go to the chapter *List of materials*.

7.2 Disassemble the RC car

At this point in the building process, the RC car had to be disassembled with the purpose of understanding the internal electronic architecture. Thanks to this previous analysis, a proper understanding of the functionality of each component was possible.

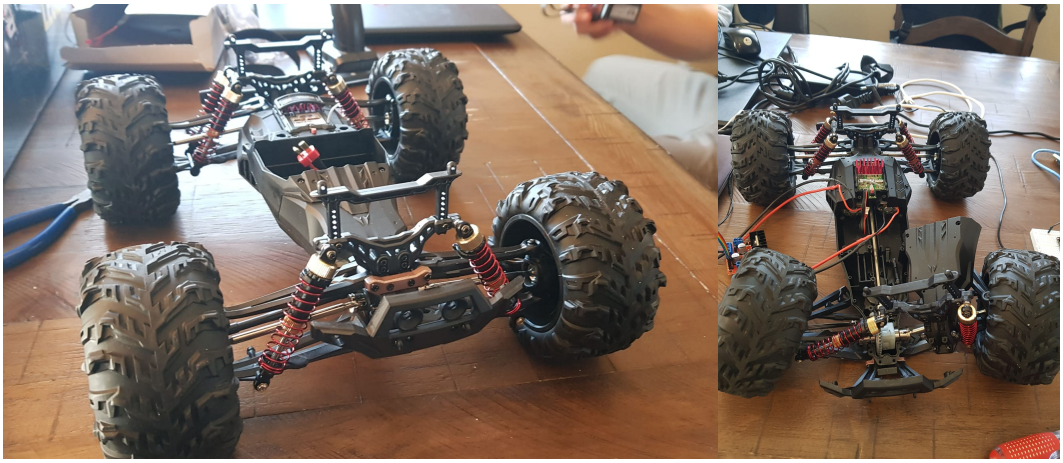


Figure 27: Disassemble of the RC car.

Regarding the electric circuit, some modifications took place. In order to run the first tests, the battery was disconnected from the original spot and reconnected to the *L982N* driver module. Another important change to mention is that the cables form the DC motors were elongated and reconnected to the motor's driver module.

7.3 Design of a 3D support

A support base for all the components was of relevant importance. For doing so, the following prototype was designed:

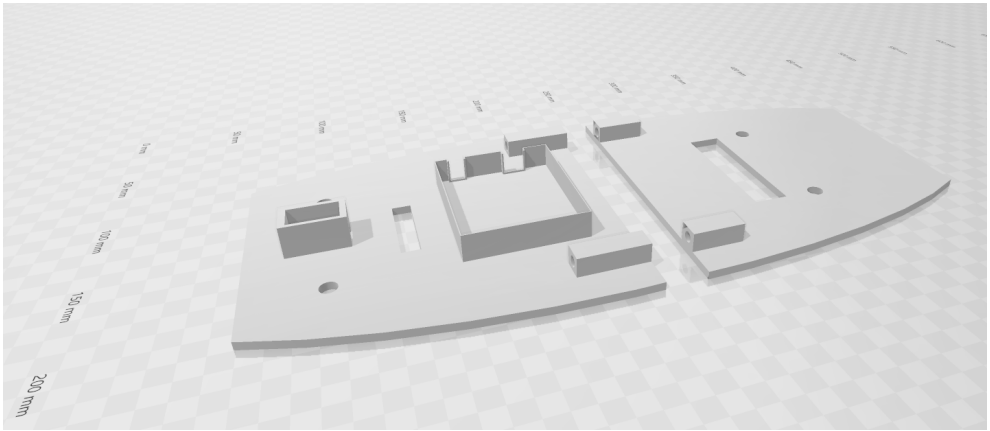


Figure 28: Design of a support base using *SketchUp 3D Software.*

Regarding the printing process of Figure 28, the piece was not possible to be printed all at once and was printed in two parts because of the printer's base size. As can be seen, the support includes a hollow for the *Arduino UNO* board and the servo.

7.4 Initial connections in the breadboard and first test of the code

Before welding the final connections, test first the Arduino code was important. Using a breadboard, the code was checked using the Wi-Fi module and smaller DC motors.

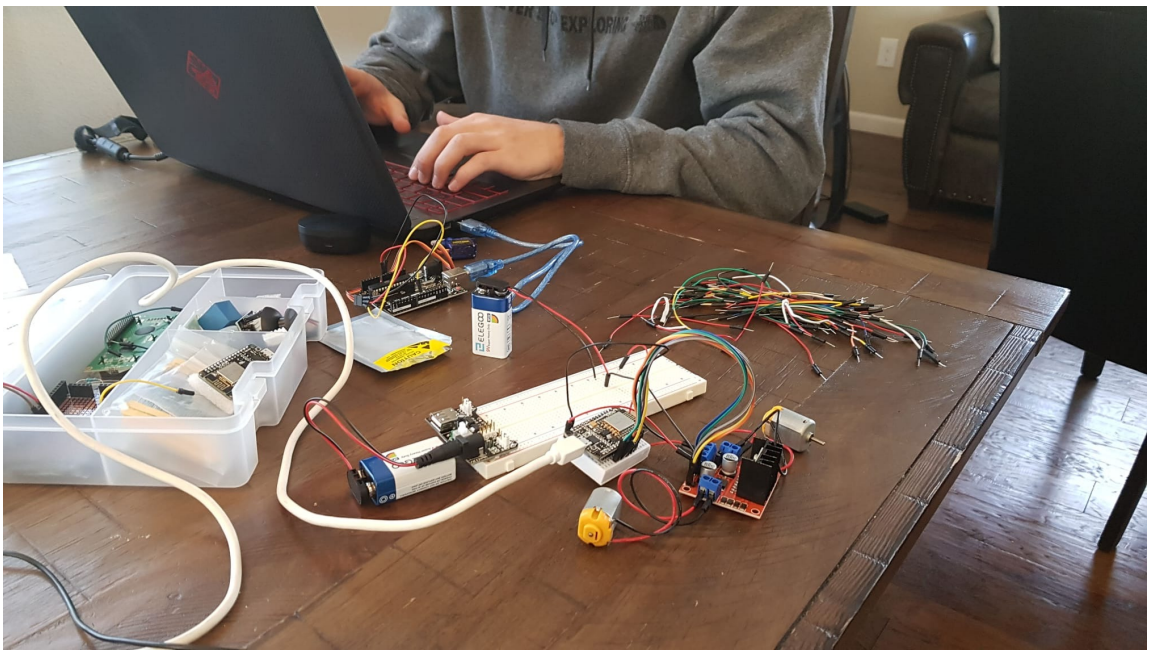


Figure 29: First testing of the code using a breadboard.

7.5 Weld the components to the PCB circuit board

Due to organizational reasons, all the wires and components were weld to a PCB. This decision was taken also for space and efficiency purposes. At this stage, the PCB circuit board looked like that:

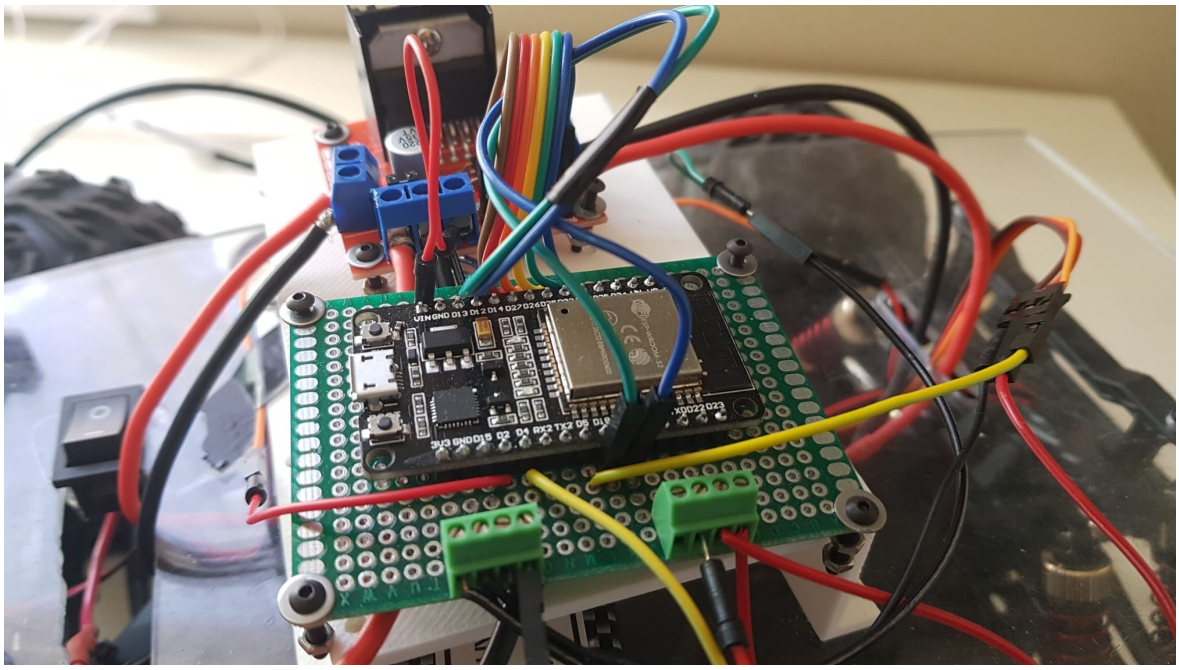


Figure 30: PCB circuit board

7.6 Final base design and 3D supports

In order to fit the battery of the robot, the following support was designed:

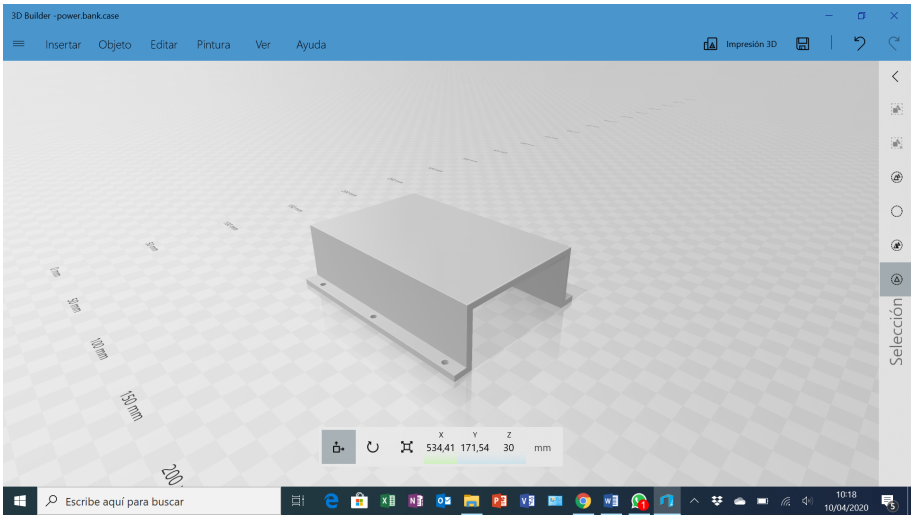


Figure 31: Battery support design.

As it was explained in previous chapters, a smartphone was used as an IP camera. Support for protecting the device was designed:

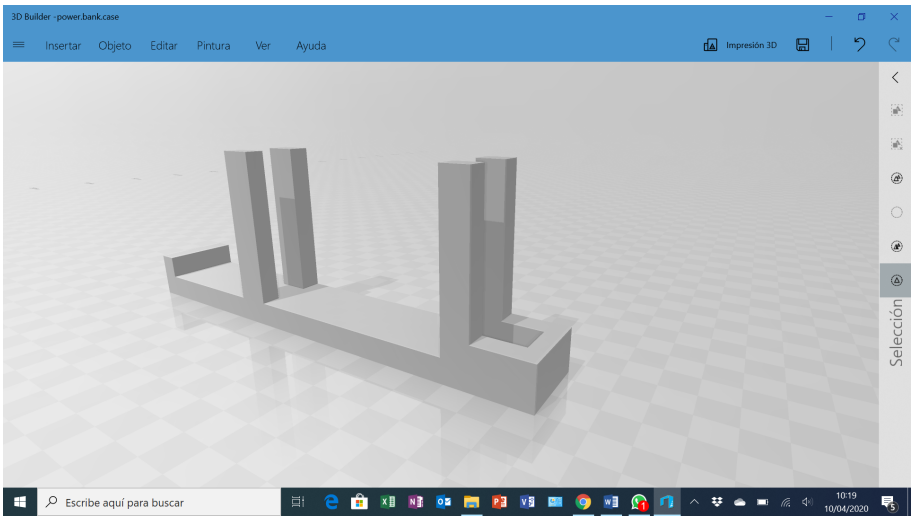


Figure 32: Smartphone support design.

The servomotor needed also a cavity. In order to guarantee its locomotion, the next image is the same support from figure 32 from the bottom.

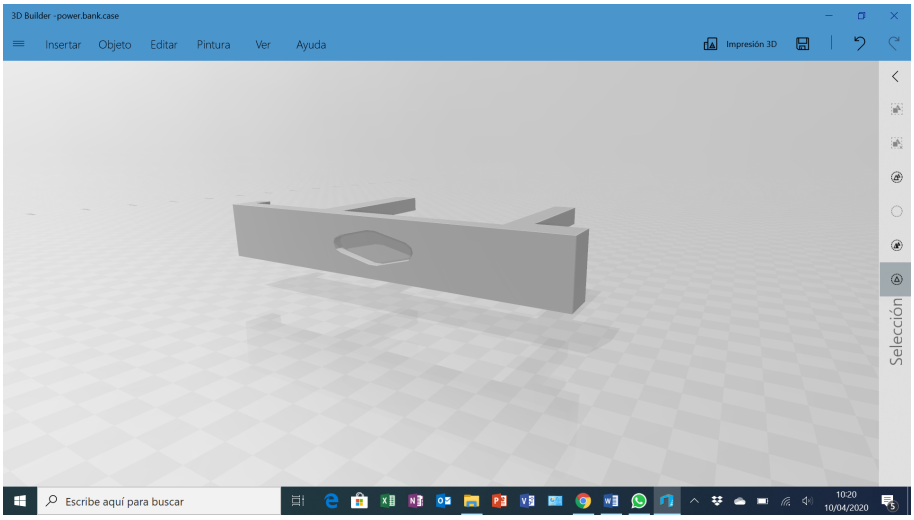


Figure 33: Servo motor support design.

Once the project evolved, it was found experimentally that the printed base support was not that stable as was thought. A new idea popped up where a methacrylate plastic base with more rigidity was able to sustain all the components using only once a structural piece instead of two. At this stage, the final shape of the robot was the following:

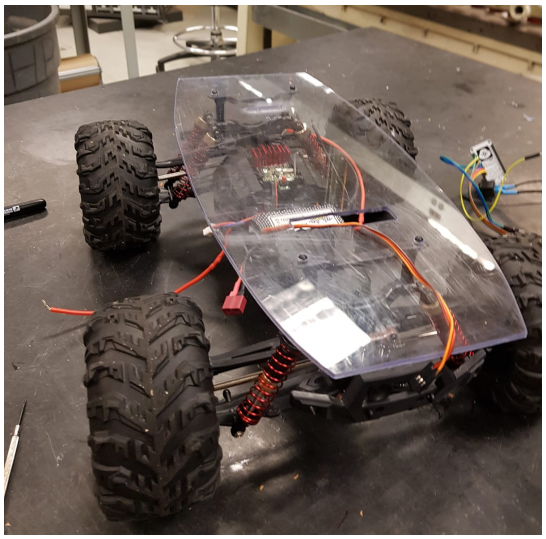


Figure 34: Usage of a methacrylate plastic base.

7.7 Assembling

When every part was properly designed and tested, the assembly took place. After checking there was no short-circuits problems in the electrical system, the robot was finally tested. At this point, the robot had the next appearance:

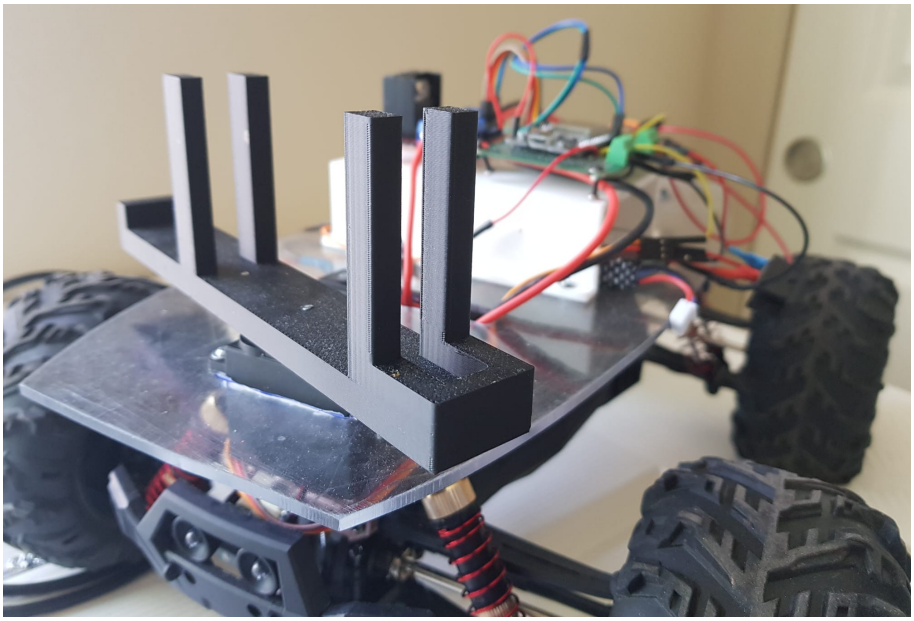


Figure 35: Assembling process.

7.8 Electrical diagram

The scheme that summarize the electrical diagram was done:

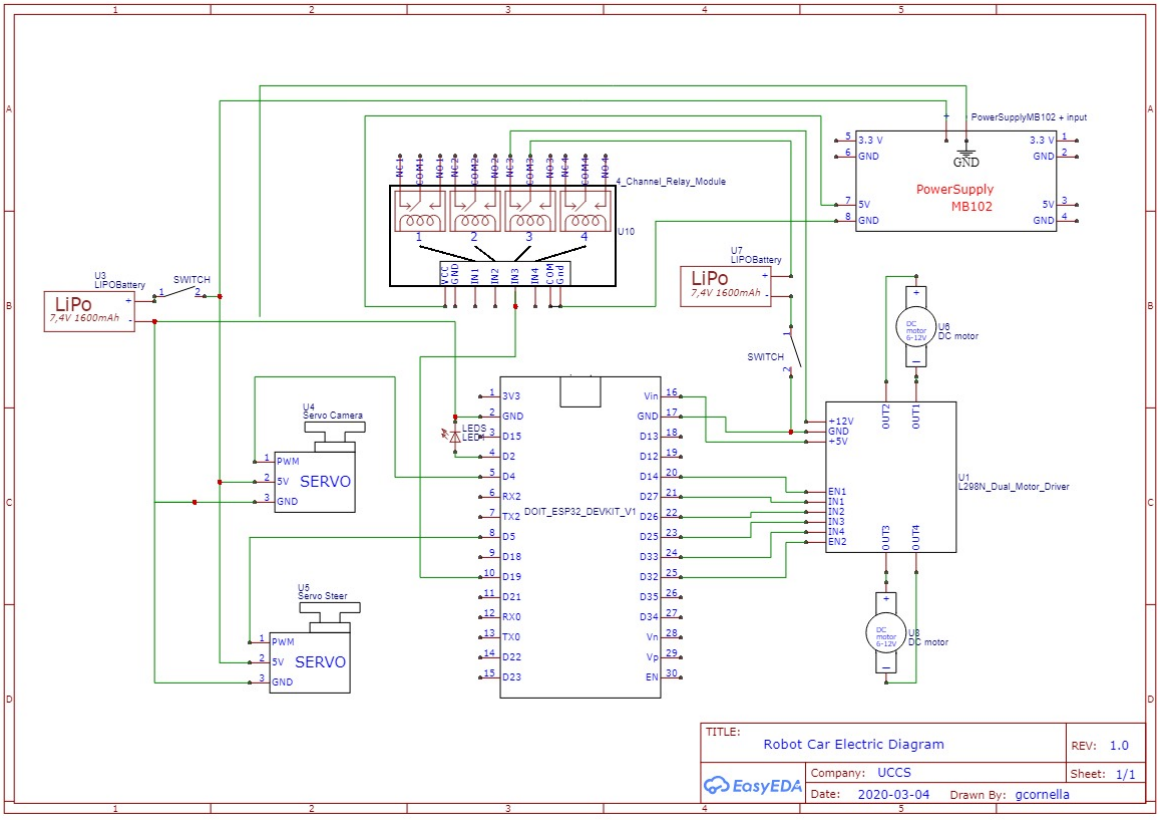


Figure 36: Electrical diagram.

7.8.1 Analysis of the servo motors

At this stage of the project, an analysis of the current that flowed across the servos was done. For obtaining its values, an Arduino program was coded using the analog pins. The servos were connected to them and a voltage fall was detected in four resistors of value 10 Ω.

Always bearing in mind whenever the domain of degrees of the servo motors, some test programs were used. As an example, the following code shows an iterative loop that increased the servo's degrees and after calculated the mean current consumption. In a physical way, the next conversion was applied for converting an analog value into current:

$$Resistance = \frac{1}{\frac{1}{R1} + \frac{1}{R2} + \frac{1}{R3} + \frac{1}{R4}} = \frac{10}{4} \tag{1}$$

As it is already known:

$$R1 = R2 = R3 = R4 = 10\Omega$$

Therefore, the current is:

$$I = \frac{5V}{1023(analog)} \cdot V(analog) \cdot \frac{4}{10}(\frac{1}{\Omega}) \cdot \frac{1000mA}{1A} \tag{2}$$

The following code was written:

```
for (int servo1Position = 20; servo1Position < 160;
     servo1Position += 2){
    t++;
    servo1.write(servo1Position);
    delay(200);
    voltageVal1 = analogRead(analogPin0);
    currentVal1=((5.0*voltageVal1)/1023.0)*
                (4.0/10.0)*1000.0;
    c1+=currentVal1;
    mean1=c1/(t*2);
    servo2Position+=1;
    servo2.write(servo2Position);
    delay(200);
    voltageVal2 = analogRead(analogPin1);
    currentVal1=((5.0*voltageVal1)/1023.0)*
                (4.0/10.0)*1000.0;
    c2+=currentVal2;
    mean2=c2/t;
    sum=mean1+mean2;
```

Listing 26: how to convert from an analog value into current)

For understanding the current consumption of the servos, two plots were obtained: with and without a load in the servo that controlled the audiovisual devices.

7.8.1.1 Servo's current consumption without load: Because of the presence of an excessive number of pikes, an average value was also plotted by the program $mean_sum$. It can be seen that the mean current consumption had a value of around 150 mA in the case of both servos when they were running at the same time. Once they stopped, the current consumption reduced at an approximate value of 6 mA.

Another important value to analyze is the 600 mA peak, which was nothing that the LI-PO battery could not supply. Nonetheless, the mean value was required for obtaining consumption.

With detail in the first part of the graph, an initial current peak can be observed when powering occurs for the first time. The reason why this happens is because of inertia reasons when starting. In a stationary state, only its armature resistance limits the current flow. Once the motor starts functioning, an opposite power supply is generated. This fact explains that when a servo makes small movements, higher peaks can be observed in contrast with moving the servos a larger distance. In the latter case, the effective voltage is less and current drops.

Figure 37: Servo's current consumption without load.

7.8.1.2 Servo's current consumption with load: Once the cameras were added to the system, some changes were detected:

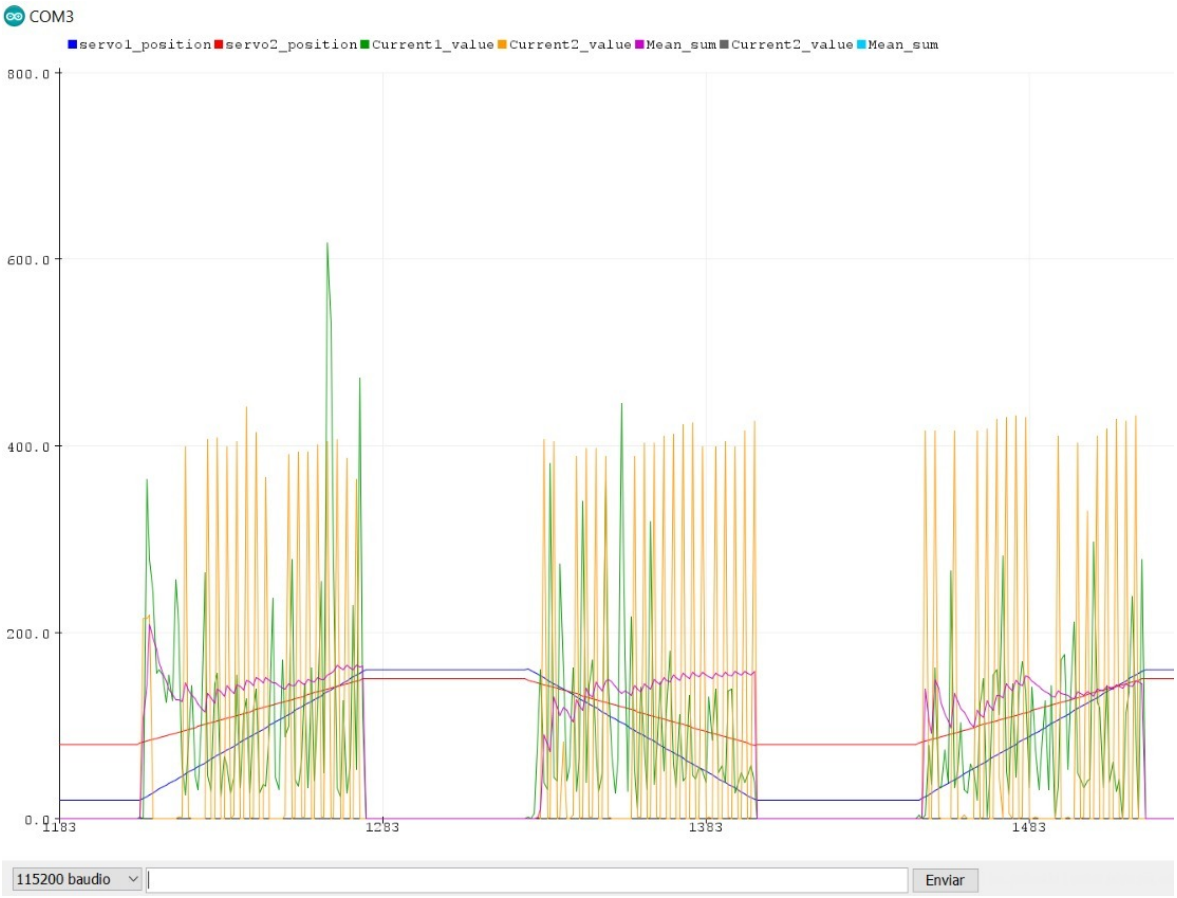


Figure 38: Servo's current consumption with loads.

The current consumption was detected to increase up to a 175 mA. Although, the steering servo was not tested with load because of difficulties moving the wheels in a stationary state, it was estimated to reach 200 mA values.

7.8.2 Current analysis of the DC motors

Due to the incapability of obtaining the DC motors current consumption, the Arduino *Serial Plotter* was used again with the following current estimation using analog values:

$$I = \frac{\frac{AnalogValue}{1024} \cdot 5000mV - OffsetValue(mV)}{Sensitivity(mV/A)} \tag{3}$$

For this purpose, the following code was made to print the values in the Serial Plotter:

```
// Variables used and declaration of values
const int currentPin = A0;
int sensitivity = 100; //it is a 20A type sensor
int adcValue= 0;
int offsetVoltage = 2500;
double adcVoltage = 0;
double currentValue = 0;

// Then the calculations
adcValue = analogRead(currentPin);
adcVoltage = (adcValue / 1024.0) * 5000;
currentValue = ((adcVoltage-offsetVoltage)/sensitivity);

Serial.println(currentValue);
```

Listing 27: Program to calculate the current from DC motors.

7.8.2.1 Current analysis of the DC without friction: An overheat of the sink in the first prototype was detected. Doing a consummation study it was seen that the motors when were stopped it consumes about 1000 mA. To sort out this problem, in the final project a relay was added. This relay cut the power supply of the motor driver *L982N* when the motors were stopped. As illustrated in figure 39 there are four relevant aspect to discuss.

- When the joystick of the Xbox Controller is in the center, the current is 0 mA so the motors are stopped.
- When the motors turn on for the first time it makes a 5 A pick. That is another reason that produces a heat of the sink because the L982N sink allows a great dissipation until 3 A. So, to improve this project is recommended to use the *L923D* motor driver.
- When the motors turn on for the second time due to the wheels' inertia because there is no friction the current necessary to start is less than 5 A, specifically 3 A.

- The mean value of the current drawn when the motors are running at maximum speed is 1100 mA.

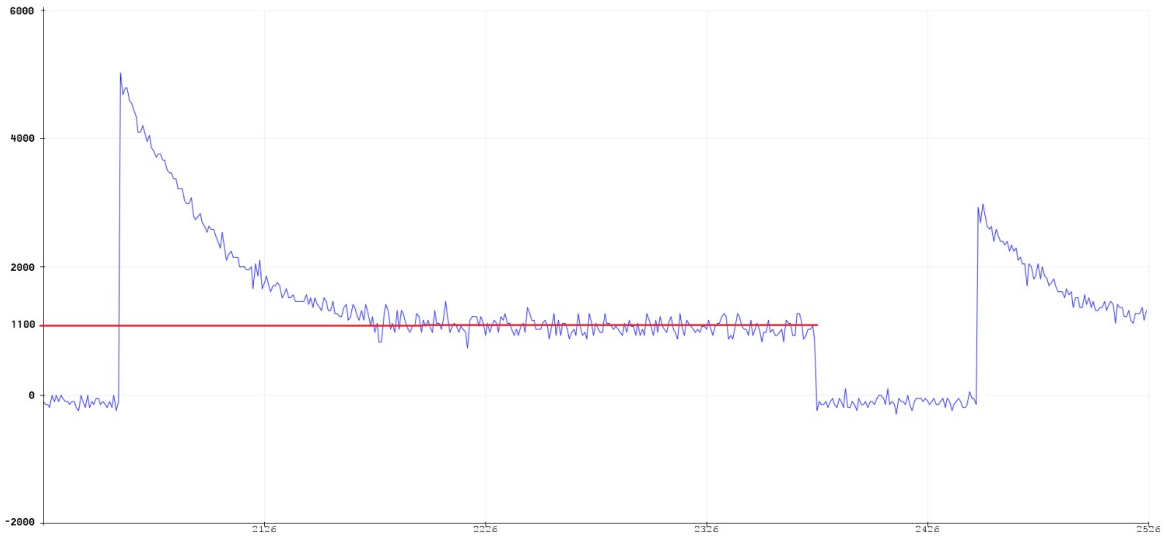


Figure 39: Motor's current consumption without friction.

7.8.2.2 Current analysis of the DC with friction: Unlikely for the analysis, there is no data obtained with the DC motors with load. Because the robot could not be in touch with the floor during the serial data measurement with Arduino, this study was not possible. However, it is estimated that this consumption is higher if the terrain and the elevation are modified.

7.8.3 Current consumption

The total consumption of the robot was 1.34 A when the robot had no friction but when it comes to the friction, the current estimation was about 2.79 A but it depended on the type of the terrain and the slop.

	No friction/ No load	**Friction/ load**
Servomotors	150 mA	175-200 mA
DC motors	1100 mA	2000-2500 mA
LED	20 mA x 2 LEDS	20 mA x 2 LEDS
ESP32	50 mA	50 mA
TOTAL MAX	1340 mA	2790 mA

Table 6: Power study of the robot.

7.9 Final testing

Figure 40: Upper picture of the final testing.

Figure 41: Frontal picture of the final testing.

8 Problems faced & solutions

- **Steer's servo:** To control the steer of the robot the *SG90* servomotor was used. The servo originally from the RC car *LG-ZJ04* could not be implemented because it has five wires and the *"servo.h"* Arduino's library can only control a servo with three wires. So, it was opted to disassemble the original servo and fit the i inside the casing of the *LG-ZJ04* (see page 14).

- **Power supply:** In order to power the robot a power bank of 5V and 2000 mAh was chosen. The main problem was that the power bank can not supply low currents and when the servomotors stop, the servos consume 6 mA and while they are turning they consume 150 mA. So, when the power bank had to provide only 6 mA, due to the internal circuit it could not supply it (see pag. 15).

- **Wi-Fi module:** Firstly, the *ESP8266* module was used. Seeing that this module did not have sufficient pins to control all the outputs, the *ESP32* Wi-Fi module was bought. The *ESP8266* has seven outputs otherwise the *ESP32* has more than twenty output pins and the project needed nine outputs to control the robot (see page 16).

- **University's Wi-Fi:** The biggest problem of the project was how to connect to the university network with the Wi-Fi module and the camera. None of the three networks could suit the project. The security of *Eudoram* and *UCCS-Wireless* was *WPA2 enterprise* and the *ESP32* can not connect to it. In regard to the *UCCS-Guest*, the camera can not send the video through the router ports because of the university's policies not allow it. To sort out the problem, the IT department offered to integrate another wireless network only for the project but due to the COVID-19, the project had to be finished at home (see page 23).

- **Cameras:** A few cameras were tested before chose the camera of the cell phone. The *AMCREST IP camera* was bought, nevertheless, it did not connect to the university's Wi-Fi since it did not support WPA2-Enterprise. After that, the *Cisco Surveillance IP Camera 2500* was bought but when it arrived some pieces were missing. Seeing that, it was returned. So finally because all the people have a cell phone and it connects to the university network, the camera of the cell phone was used. This change allowed to reduce the cost of the project. (see page 27)

- **Overheating of the sink:** A high heat of the sink of the motor driver L982N was experimented. This heat was due to the initialize current when the motors turn on because a 5 A peak of current was produced and the motor driver only allows 3 A. So, the L923D motor driver is recommended to replace the L982N (see page 53).

9 Conclusion

In this project, an application to control a robot through Wi-Fi has been developed. This robot is commanded via the Xbox controller. In order to do the application a C# language has been used. To be able to communicate the app with the robot via Wi-Fi, an ESP32 module has been implemented in the car. This Wi-Fi module has been programmed with the Arduino language. The robot also contains an IP camera whereupon its operator can access live to the robot's vision through the application.

The main goal of this project was to be able to create a project that could be implemented in a semester course at the University of Colorado Colorado Springs. After thoroughly reading this report and having the full project into GitHub [21], we believe that it is possible to recreate it if the students already have notions of C# and Arduino. This is a useful project to go deep into creating Windows applications and internet communications protocols (UDP vs TCP).

Out of the scope of this project and a wonderful idea to amplify this project could be to implement an algorithm to detect objects while the car is driven. Another thought could be to extract data from the robot with sensors and print it to the app, like the Rovers (NASA robot) does.

References

[1] Boston Dynamics. *Spot*. https://www.bostondynamics.com/spot. 2020.

[2] NASA. *Mars Exploration Rovers*. https://mars.nasa.gov/mer/mission/rover/eyes-and-senses/. 2019.

[3] BuiltIn. *12 examples of rescue robots*. https://builtin.com/robotics/rescue-robots. 2020.

[4] Legendary. *390 MOTOR - PART NUMBER LG-DJ01*. https://laegendary.com/products/rc-car-390motor-includes-gear-accessory-spare-parts-25-dj01-for-legend-rc-car. Feb. 2020.

[5] Laegendary. *ESC ELECTRONIC SPEED CONTROLLER*. https://laegendary.com/collections/rc-car-parts/products/rc-car-electronic-speed-controller/-assembly-accessory-spare-parts-25-zj07-for-legend-rc-car. 2020.

[6] How to mechatronics. *Arduino DC Motor Control Tutorial – L298N — PWM — H-Bridge*. https://laegendary.com/products/rc-car-390motor-includes-gear-accessory-spare-parts-25-dj01-for-legend-rc-car. Feb. 2020.

[7] Mechatronics. *Arduino DC Motor Control Tutorial*. https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/. 2020.

[8] AI Shack. *Positioning a servo*. https://aishack.in/tutorials/servo-motors/. 2020.

[9] Components 101. *MG996R Servo Motor*. https://components101.com/motors/mg996r-servo-motor-datasheet. Mar. 2020.

[10] Tower Pro. *SERVO MOTOR SG90 - Datasheet*. http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf. Mar. 2020.

[11] Legendary. *5 WIRES SERVO - PART NUMBER LG-ZJ04*. https://laegendary.com/products/rc-car-5-wires-servo-accessory-spare-parts-25-zj04-for-legend-rc-car. Mar. 2020.

[12] Legendary. *1600 MAH 7.4V 2S 25C LI-PO RECHARGEABLE BATTERY*. https://laegendary.com/collections/rc-car-parts/products/rc-cars-1pc-7-4v-1600mah-25c-t-connector-li-polymer-rechargeable-battery-for-/legend-high-speed-remote-control-truck-accessory-supplies. Apr. 2020.

[13] ESPRESSIF. *ESP32 overview*. https://www.espressif.com/en/products/hardware/esp32/overview. Jan. 2020.

[14] Espressif Systems. *ESP32 Datasheet*. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. 2020.

[15] Random Nerds Tutorial. *Getting Started with the ESP32 Development Board*. https://randomnerdtutorials.com/getting-started-with-esp32/. Feb. 2020.

[16] Micrium. *TCP/IP Protocol Stack*. https://www.micrium.com/iot/internet-protocols/. 2019.

[17] Wikipedia. *TCP/IP Protocol*. https://ca.wikipedia.org/wiki/Transmission_Control_Protocol. Mar. 2020.

[18] MUVI. *UDP(User Datagram Protocol)*. https://www.muvi.com/wiki/udpuser-datagram-protocol.html. 2020.

[19] UCCS. *UCCS Wireless*. https://wireless.uccs.edu/. 2020.

[20] OctoPrint. *Using an Android phone as a webcam*. https://github.com/OctoPrint/OctoPrint/wiki/Using-an-Android-phone-as-a-webcam. Oct. 2020.

[21] Guillem Cornella and Eudald Sangenís. *WIFI Robot Controlled Through Xbox Controller*. https://github.com/eudald-sangenis/WIFI_Robot_Controlled_Xbox. Apr. 2020.

A C# Code

A.A MainWindow.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using VisioForge.Types.OutputFormat;
using OpenJigWare;
using System.Net.Sockets;
using System.Text;
using System.Timers;

namespace WIFI_Robot_Controlled_Xbox
{
    /// <summary>
    /// Install NuGet Pakages:
    ///     - VisioForge
    ///     - Material Theme
    ///     - Sharp DX
    ///     - Sharp DX Input
    ///     - OpenJigWare
    /// </summary>

    public partial class MainWindow : Window
    {
        /// variables int
        int count_btn_snap = 0;
        int count_btn_recd = 0;
        /// variables bool
        bool connect_cam;
        bool XBOX_X;
        bool start_pilot;
        bool lights_bool;
        /// variables string
        string leds;
        string jx0;
        string jx1;
        string jy1;

        ///create a new UDP object in the port 2000
        UdpClient udpClient  = new UdpClient(2000);
        ///create a new timer object
        private static Timer timer;
```

```
public MainWindow()
{
    InitializeComponent();

    ///Set up timer
    timer = new Timer();
    timer.Interval = 40;
    timer.Elapsed += new ElapsedEventHandler(this.
    timer1_Tick_1);
    timer.Enabled = true;
}

//-----------------------------
//    CAMARA'S CONTROLS
//-----------------------------
#region CAMARA'S CONTROLS
///Camara connection
private void Connect(object sender, RoutedEventArgs e)
{
    if(Global_Variables.ip_cam == null || Global_Variables.
    port_cam == 0)
    {
        MessageBox.Show("IP Camera or Port Camera isn't
        fulfilled.");
    }
    ///Error - Port number overflow
    else if (Global_Variables.port_cam > 65536)
    {
        MessageBox.Show("Camera's Port number is overflowed
        ." + "\n" + "It has to be smaller than 65536");
    }
    ///Error - IP number overflow
    else if (Global_Variables.ip_cam.Length > 15)
    {
        MessageBox.Show("Camera's IP number is overflowed."
        + "\n" + "It has to be smaller than 15 digits");
    }
    ///Error - First fill the Ip Configuration:
    else
    {
        ///initialize the image
        videoCapture1.IP_Camera_Source = new
        VisioForge.Types.Sources.IPCameraSourceSettings()
        {
            ///Especific URL to connect to S8
            URL = "http:/" + Global_Variables.ip_cam + ":"
```

```
                + Global_Variables.port_cam.ToString() +
                "/video",

                ///Type HTTP: we obtein the image through and
                ///URL, LowLatency: we want the image in real
                ///time, but we lost resolution.

                Type = VisioForge.Types.VFIPSource.
                HTTP_MJPEG_LowLatency
            };
            ///initialize the audio
            videoCapture1.Audio_PlayAudio = videoCapture1.
            Audio_RecordAudio = false;
            videoCapture1.Mode = VisioForge.Types.
            VFVideoCaptureMode.IPPreview;

            connect_cam = true;
            videoCapture1.Start();
        }
    }

    ///Camara disconnection
    private void Disconnect(object sender, RoutedEventArgs e)
    {
        connect_cam = false;
        videoCapture1.Stop();
        XBOX_X = false;
    }

    ///Camera record
    private void Record(object sender, RoutedEventArgs e)
    {
        if (connect_cam == false)
        {
            MessageBox.Show("First Connect the Camera");
        }
        else
        {
            ///number of record button clicks
            count_btn_recd++;

            ///if the streaming is enable it close the object
            ///videoCapture1
            if (videoCapture1.IsEnabled)
            {
                videoCapture1.Stop();
```

```
            }

            ///Create a new videoCapture1 object
            videoCapture1.IP_Camera_Source = new VisioForge.
            Types.Sources.IPCameraSourceSettings()
            {
                URL = "http:/" + Global_Variables.ip_cam + ":"
                + Global_Variables.port_cam.ToString() +
                "/video",

                Type = VisioForge.Types.VFIPSource.
                HTTP_MJPEG_LowLatency
            };
            ///Audio Settings
            videoCapture1.Audio_PlayAudio = videoCapture1.
            Audio_RecordAudio = false;
            ///save the video to myvideos folder with the name
            ///of vid_{count_btn_recd}.mp4
            videoCapture1.Output_Filename = Environment.
            GetFolderPath(Environment.SpecialFolder.
            MyVideos) + $"\\vid_{count_btn_recd}.mp4";
            ///Output video format
            videoCapture1.Output_Format = new VFWMVOutput();
            ///Type of video
            videoCapture1.Mode = VisioForge.Types.
            VFVideoCaptureMode.IPCapture;

            videoCapture1.Start();
        }
    }

    ///Camara snapshoot
    private void Snapshot(object sender, RoutedEventArgs e)
    {
        if (connect_cam == false)
        {
            MessageBox.Show("First Connect the Camera");
        }
        else
        {
            ///number of snapshot button clicks
            count_btn_snap++;
            ///save the snapshot to mypictures folder with the
            ///name of frame_{count_btn_snap}.jpg
            videoCapture1.Frame_Save(Environment.GetFolderPath
            (Environment.SpecialFolder.MyPictures) + $"\\frame_
```

```
            {count_btn_snap}.jpg", VisioForge.Types.
            VFImageFormat.JPEG, 85);
        }

    }
    #endregion CAMARA'S CONTROLS



    //------------------------------
    //     IP CONFING WINDOW
    //------------------------------
    #region IP CONFING WINDOW
    private void Ip_config_Click(object sender,
    RoutedEventArgs e)
    {
        IP_Window iP_Window = new IP_Window();
        iP_Window.Show();
    }
    #endregion IP CONFING WINDOW



    //------------------------------
    //          Joystick
    //------------------------------
    #region Joystick

    ///Joystick Declaration
    private Ojw.CJoystick m_CJoy = new Ojw.CJoystick(
    Ojw.CJoystick._ID_0);

    ///Timer to periodically check joystick connection
    private Ojw.CTimer m_CTmr_Joystick = new Ojw.CTimer();

    //analize data and send it
    #region Joystick data
    private void FJoystick_Check_Data()
    {
        #region Joystick Check Data
        #region arrows
        ///leds off - arrow left
        if (m_CJoy.IsDown(Ojw.CJoystick.PadKey.POVLeft))
        {
            leds = "2";
        }
```

```
///leds on - arrow down
else if (m_CJoy.IsDown(Ojw.CJoystick.PadKey.POVDown) ==
true)
{
    leds = "1";
}
///maintain the led status
else
{
    leds = "0";
}
#endregion arrows

#region buttons
// Button A
if (m_CJoy.IsDown(Ojw.CJoystick.PadKey.Button1) ==
true)
{
    ///Error - Null values ip_rob or port_rob
    if (Global_Variables.ip_rob == null ||
    Global_Variables.port_rob == 0)
    {
        MessageBox.Show("IP Robot or Port Robot isn't
        fulfilled.");
    }
    ///Error - Port number overflow
    else if (Global_Variables.port_rob > 65536)
    {
        MessageBox.Show("Robot's Port number is
        overflowed." + "\n" + "It has to be
        smaller than 65536");
    }
    ///Error - IP number overflow
    else if (Global_Variables.ip_rob.Length > 15)
    {
        MessageBox.Show("Robot's IP number is
        overflowed." + "\n" + "It has to be
        smaller than 15 digits");
    }
    ///Error - First fill the Ip Configuration:
    else
    {
        ///To be able to send the joysitcks
        ///value data to the robot
        start_pilot = true;
    }
```

```
            }
            // Button X
            if (m_CJoy.IsDown_Event(Ojw.CJoystick.PadKey.Button3)
            == true && XBOX_X == false)
            {
                this.Dispatcher.Invoke(() =>
                {
                    BtnConnect.RaiseEvent(new RoutedEventArgs(
                    Button.ClickEvent));
                });
                XBOX_X = true;
            }
            // Button Y
            if (m_CJoy.IsDown_Event(Ojw.CJoystick.PadKey.Button4)
            == true)
            {
                this.Dispatcher.Invoke(() =>
                {
                    BtnDisconnect.RaiseEvent(new RoutedEventArgs(
                    Button.ClickEvent));
                });
            }
            #endregion buttons

            #region R & L buttons
            //Button LB -- Photo
            if (m_CJoy.IsDown_Event(Ojw.CJoystick.PadKey.Button5)
            == true)
            {
                this.Dispatcher.Invoke(() =>
                {
                    BtnSnap.RaiseEvent(new RoutedEventArgs(Button.
                    ClickEvent));
                });
            }
            //Button RB -- Record
            if (m_CJoy.IsDown_Event(Ojw.CJoystick.PadKey.Button6)
            == true)
            {
                this.Dispatcher.Invoke(() =>
                {
                    BtnRecord.RaiseEvent(new RoutedEventArgs(Button
.
                    ClickEvent));
                });
            }
```

```
#endregion R & L buttons


#region Joysticks values
//DATA TO SEND:
//camera servo
jx0 = Math.Round(m_CJoy.dX0, 1).ToString();
//steer  servo
jx1 = Math.Round(m_CJoy.dX1, 1).ToString();
//velocity and direction
jy1 = (1-Math.Round(m_CJoy.dY1, 1)).ToString();

jx0 = jx0.Replace(",", ".");
jx1 = jx1.Replace(",", ".");
jy1 = jy1.Replace(",", ".");

if (jx0 == "0") jx0 = "0.0";
if (jx1 == "0") jx1 = "0.0";
if (jy1 == "0") jy1 = "0.0";
if (jx0 == "1") jx0 = "1.0";
if (jx1 == "1") jx1 = "1.0";
if (jy1 == "1") jy1 = "1.0";

//DATA TO SHOW TO THE SCREEN
var camera_tilt = (Math.Round(m_CJoy.dX0, 1) - 0) *
(170 - 10) / (1 - 0) + 10;
var steer = (Math.Round(m_CJoy.dX1, 1) - 0)*(110 - 70)
/ (1 - 0) + 70;

var velocity = 0.0;
var lights = "";
var drive = "";

if((1 - Math.Round(m_CJoy.dY1, 1)) >= 0.5)
velocity = ((1 - Math.Round(m_CJoy.dY1, 1)) - 0.5 - 0)
* (100 - 0) / (0.5 - 0) + 0;

else
velocity = ((1 - Math.Round(m_CJoy.dY1, 1)) - 0.5 - 0)
* (100 - 0) / (0.5 - 0) + 0;

if (leds == "1")
{
    lights = "on";
    lights_bool = true;
}
```

```
if (leds == "2")
{
    lights = "off";
    lights_bool = false;
}
if (leds == "0")
{
    if (lights_bool == true) lights = "on";
    else lights = "off";
}
if (start_pilot == true) drive = "Available";
else drive = "Disable";
#endregion Joysticks values


//SEND DATA & RECIVE
#region Send & print data

this.Dispatcher.Invoke(() =>
{
    if (start_pilot == true)
    {
        //send data to arduino
        var joystick_values = "jx0" + jx0 + "jx1" + jx1
        + "jy1" + jy1 + "LED" + leds;

        udpClient.Connect(Global_Variables.ip_rob,
        Global_Variables.port_rob);

        Byte[] sendBytes_jx0 = Encoding.ASCII.GetBytes(
        joystick_values);

        udpClient.Send(sendBytes_jx0, sendBytes_jx0.
        Length);
    }

    //print data
    lb_joysticks.Text = "  Camera: " + camera_tilt +
    " degrees" + "\n\n" + "  Steer: " + steer +
    " degrees \n\n"  + "  Velocity: " + velocity;

    lb_buttons.Text = "  Drive: " + drive + "\n\n" +
    "  Lights: " + lights;
});
#endregion Send & print data
#endregion Joystick Check Data
```

```
        }
        #endregion Joystick data

        //timer event
        #region timer event
        public void timer1_Tick_1(object sender, EventArgs e)
        {
            // update joystick information
            m_CJoy.Update();
            // Joystick Data Check
            FJoystick_Check_Data();
        }
        #endregion timer event

        #endregion Joystick
    }
}
```

A.B MainWindow.xaml

```
<Window
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
    presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend
    /2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
    compatibility/2006"
        xmlns:local="clr-namespace:WIFI_Robot_Controlled_Xbox"
        xmlns:WPF="clr-namespace:VisioForge.Controls.UI.WPF;
    assembly=VisioForge.Controls.UI"
        x:Class="WIFI_Robot_Controlled_Xbox.MainWindow"
        mc:Ignorable="d"
        Title="WIFI RC ROBOT" Height="515" Width="1000" ResizeMode
    ="NoResize" Background="Gainsboro">

     <!--Outside Grid-->
     <Grid x:Name="OutsideGrid">
         <!--Work Space Definition-->
         <Grid.ColumnDefinitions>
             <ColumnDefinition Width="15"/>
             <ColumnDefinition Width="450"/>
             <ColumnDefinition Width="*"/>
             <ColumnDefinition Width="15"/>
         </Grid.ColumnDefinitions>
         <Grid.RowDefinitions>
             <RowDefinition Height="10"/>
             <RowDefinition Height="336"/>
             <RowDefinition Height="*"/>
             <RowDefinition Height="10"/>
         </Grid.RowDefinitions>

         <!--Camara Grid-->
         <Grid x:Name="InnerCameraGrid" Grid.Column="1" Grid.Row
    ="2">
             <Grid.ColumnDefinitions>
                 <ColumnDefinition Width="20"/>
                 <ColumnDefinition Width="*"/>
                 <ColumnDefinition Width="*"/>
                 <ColumnDefinition Width="20"/>
             </Grid.ColumnDefinitions>
             <Grid.RowDefinitions>
                 <RowDefinition Height="20"/>
                 <RowDefinition Height="50"/>
```

```
                <RowDefinition Height="50"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>

            <!--Buttons Camara-->
            <Button x:Name="BtnConnect" Click="Connect" Content="
Connect"
                Height="35" Width="140"
                Grid.Column="1" Grid.Row="1">
            </Button>
            <Button x:Name="BtnDisconnect" Click="Disconnect"
Content="Disconnect"
                Height="35" Width="140"
                Grid.Column="2" Grid.Row="1">
            </Button>
            <Button x:Name="BtnRecord" Click="Record" Content="
Record"
                Height="35" Width="140"
                Grid.Column="1" Grid.Row="2">
                <Button.ContentTemplate>
                    <DataTemplate>
                        <Image Source="C:\Users\carle\Documents\
GitHub\WIFI_Robot_Controlled_Xbox\rec.png" VerticalAlignment="
Center" Height="50"/>
                    </DataTemplate>
                </Button.ContentTemplate>
            </Button>
            <Button x:Name="BtnSnap" Click="Snapshot" Content="
Snapshot"
                Height="35" Width="140"
                Grid.Column="2" Grid.Row="2">
                <Button.ContentTemplate>
                    <DataTemplate>
                        <Image Source="C:\Users\carle\Documents\
GitHub\WIFI_Robot_Controlled_Xbox\snap.png" VerticalAlignment="
Center" Height="20"/>
                    </DataTemplate>
                </Button.ContentTemplate>
            </Button>
        </Grid>

        <!--Controls Robot & Camara Grid-->
        <Grid x:Name="ControlsGrid" Grid.Column="2" Grid.Row="1">
            <Grid.RowDefinitions>
                <RowDefinition Height="15"/>
                <RowDefinition Height="*"/>
```

```
            </Grid.RowDefinitions>
            <!-- Image -->
            <Image Source="C:\Users\carle\Documents\GitHub\
WIFI_Robot_Controlled_Xbox\xbox_controls.png" Stretch="Fill"
Grid.Row="1"/>
            <Label Content="Xbox Controller Commands" Grid.Row="1"
FontSize="25" VerticalAlignment="Top" HorizontalAlignment="
Center" FontWeight="Bold" Foreground="#333333"/>

            <!--IP configuration Btn-->
            <Button x:Name="BtnIP" Height="15" Width="15" Grid.
Column="0" Grid.Row="0" Click=" Ip_config_Click"
                    VerticalAlignment="Top" HorizontalAlignment="
Left" Margin="5,0,0,0"
                    ToolTipService.InitialShowDelay="500"
ToolTipService.ShowDuration="3000" ToolTipService.
BetweenShowDelay="10000" ToolTip="Configurate IP.">
                <Button.Template>
                    <ControlTemplate>
                        <Image Source="C:\Users\carle\Documents\
GitHub\WIFI_Robot_Controlled_Xbox\gear.png"/>
                    </ControlTemplate>
                </Button.Template>
            </Button>
        </Grid>

        <!-- Sensoring Grid + Leds-->
        <Grid x:Name="Sensoring" Grid.Column="2" Grid.Row="2">
            <Label Content="Joysticks information:"
HorizontalAlignment="Left"  VerticalAlignment="Top" FontStyle="
Normal" FontSize="14" Margin="65,0,10,0" FontWeight="Bold"/>
            <Label Content="Buttons information:"
HorizontalAlignment="Right" VerticalAlignment="Top" FontStyle="
Normal" FontSize="14" Margin="10,0,65,0"  FontWeight="Bold"/>

            <TextBox x:Name="lb_joysticks" VerticalAlignment="Top"
HorizontalAlignment="Left"  Margin="40,30,40,0" Height="90"
Width="200" Background="White"/>
            <TextBox x:Name="lb_buttons"   VerticalAlignment="Top"
HorizontalAlignment="Right" Margin="40,30,40,0" Height="90"
Width="200" Background="White"/>
        </Grid>


        <!--Camara Vision-->
        <Frame Grid.Column="1" Grid.Row="1" Height="336" Width
```

```
    ="450" Background="#333333">
            <Frame.Content>
                <TextBlock Text="No Signal" FontSize="20" FontStyle
    ="Italic" Foreground="#FFFFFF" HorizontalAlignment="Center"
    VerticalAlignment="Center"/>
            </Frame.Content>
        </Frame>

        <WPF:VideoCapture Name="videoCapture1" HorizontalAlignment
    ="Left" VerticalAlignment="Top"
                          Height="336"  Width="450"
                          Grid.Column="1" Grid.Row="1"/>
    </Grid>
</Window>
```

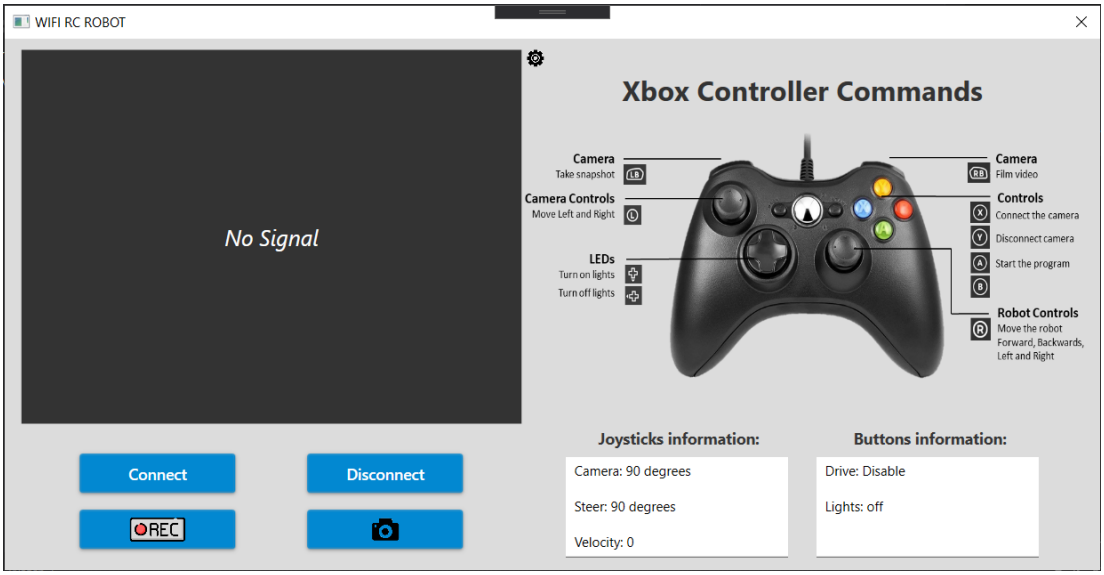


Figure 42: Xbox Controller application interface.

A.C IpWindow.xaml.cs

```
using System;
using System.Windows;

namespace WIFI_Robot_Controlled_Xbox
{
    /// <summary>
    /// Interaction logic for Window1.xaml
    /// </summary>
    public partial class IP_Window : Window
    {
        public IP_Window()
        {
            InitializeComponent();
        }

        private void Apply_Click(object sender, RoutedEventArgs e)
        {
            Global_Variables. ip_cam = txb_Ip_cam.Text;
            Global_Variables.ip_rob = txb_Ip_rob.Text;
            Global_Variables.port_cam = Convert.ToInt32(
            txb_Port_cam.Text);
            Global_Variables.port_rob = Convert.ToInt32(
            txb_Port_rob.Text);
        }
    }
}
```

A.D IpWindow.xaml

```
<Window
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/
    presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend
    /2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
    compatibility/2006"
        xmlns:local="clr-namespace:WIFI_Robot_Controlled_Xbox"
        xmlns:WPF="clr-namespace:VisioForge.Controls.UI.WPF;
    assembly=VisioForge.Controls.UI"
        x:Class="WIFI_Robot_Controlled_Xbox.MainWindow"
        mc:Ignorable="d"
        Title="WIFI RC ROBOT" Height="515" Width="1000" ResizeMode
    ="NoResize" Background="Gainsboro">

     <!--Outside Grid-->
     <Grid x:Name="OutsideGrid">
        <!--Work Space Definition-->
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="15"/>
            <ColumnDefinition Width="450"/>
            <ColumnDefinition Width="*"/>
            <ColumnDefinition Width="15"/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition Height="10"/>
            <RowDefinition Height="336"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="10"/>
        </Grid.RowDefinitions>

        <!--Camara Grid-->
        <Grid x:Name="InnerCameraGrid" Grid.Column="1" Grid.Row
    ="2">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="20"/>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="20"/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition Height="20"/>
                <RowDefinition Height="50"/>
```

```
                <RowDefinition Height="50"/>
                <RowDefinition Height="*"/>
            </Grid.RowDefinitions>

            <!--Buttons Camara-->
            <Button x:Name="BtnConnect" Click="Connect" Content="
Connect"
                Height="35" Width="140"
                Grid.Column="1" Grid.Row="1">
            </Button>
            <Button x:Name="BtnDisconnect" Click="Disconnect"
Content="Disconnect"
                Height="35" Width="140"
                Grid.Column="2" Grid.Row="1">
            </Button>
            <Button x:Name="BtnRecord" Click="Record" Content="
Record"
                Height="35" Width="140"
                Grid.Column="1" Grid.Row="2">
                <Button.ContentTemplate>
                    <DataTemplate>
                        <Image Source="C:\Users\carle\Documents\
GitHub\WIFI_Robot_Controlled_Xbox\rec.png" VerticalAlignment="
Center" Height="50"/>
                    </DataTemplate>
                </Button.ContentTemplate>
            </Button>
            <Button x:Name="BtnSnap" Click="Snapshot" Content="
Snapshot"
                Height="35" Width="140"
                Grid.Column="2" Grid.Row="2">
                <Button.ContentTemplate>
                    <DataTemplate>
                        <Image Source="C:\Users\carle\Documents\
GitHub\WIFI_Robot_Controlled_Xbox\snap.png" VerticalAlignment="
Center" Height="20"/>
                    </DataTemplate>
                </Button.ContentTemplate>
            </Button>
        </Grid>

        <!--Controls Robot & Camara Grid-->
        <Grid x:Name="ControlsGrid" Grid.Column="2" Grid.Row="1">
            <Grid.RowDefinitions>
                <RowDefinition Height="15"/>
                <RowDefinition Height="*"/>
```

```
            </Grid.RowDefinitions>
            <!-- Image -->
            <Image Source="C:\Users\carle\Documents\GitHub\
WIFI_Robot_Controlled_Xbox\xbox_controls.png" Stretch="Fill"
Grid.Row="1"/>
            <Label Content="Xbox Controller Commands" Grid.Row="1"
FontSize="25" VerticalAlignment="Top" HorizontalAlignment="
Center" FontWeight="Bold" Foreground="#333333"/>

            <!--IP configuration Btn-->
            <Button x:Name="BtnIP" Height="15" Width="15" Grid.
Column="0" Grid.Row="0" Click=" Ip_config_Click"
                    VerticalAlignment="Top" HorizontalAlignment="
Left" Margin="5,0,0,0"
                    ToolTipService.InitialShowDelay="500"
ToolTipService.ShowDuration="3000" ToolTipService.
BetweenShowDelay="10000" ToolTip="Configurate IP.">
                <Button.Template>
                    <ControlTemplate>
                        <Image Source="C:\Users\carle\Documents\
GitHub\WIFI_Robot_Controlled_Xbox\gear.png"/>
                    </ControlTemplate>
                </Button.Template>
            </Button>
        </Grid>

        <!-- Sensoring Grid + Leds-->
        <Grid x:Name="Sensoring" Grid.Column="2" Grid.Row="2">
            <Label Content="Joysticks information:"
HorizontalAlignment="Left"  VerticalAlignment="Top" FontStyle="
Normal" FontSize="14" Margin="65,0,10,0" FontWeight="Bold"/>
            <Label Content="Buttons information:"
HorizontalAlignment="Right" VerticalAlignment="Top" FontStyle="
Normal" FontSize="14" Margin="10,0,65,0"  FontWeight="Bold"/>

            <TextBox x:Name="lb_joysticks" VerticalAlignment="Top"
HorizontalAlignment="Left"  Margin="40,30,40,0" Height="90"
Width="200" Background="White"/>
            <TextBox x:Name="lb_buttons"   VerticalAlignment="Top"
HorizontalAlignment="Right" Margin="40,30,40,0" Height="90"
Width="200" Background="White"/>
        </Grid>


        <!--Camara Vision-->
        <Frame Grid.Column="1" Grid.Row="1" Height="336" Width
```

```
    ="450" Background="#333333">
            <Frame.Content>
                <TextBlock Text="No Signal" FontSize="20" FontStyle
    ="Italic" Foreground="#FFFFFF" HorizontalAlignment="Center"
    VerticalAlignment="Center"/>
            </Frame.Content>
        </Frame>

        <WPF:VideoCapture Name="videoCapture1" HorizontalAlignment
    ="Left" VerticalAlignment="Top"
                          Height="336"  Width="450"
                          Grid.Column="1" Grid.Row="1"/>
    </Grid>
</Window>
```

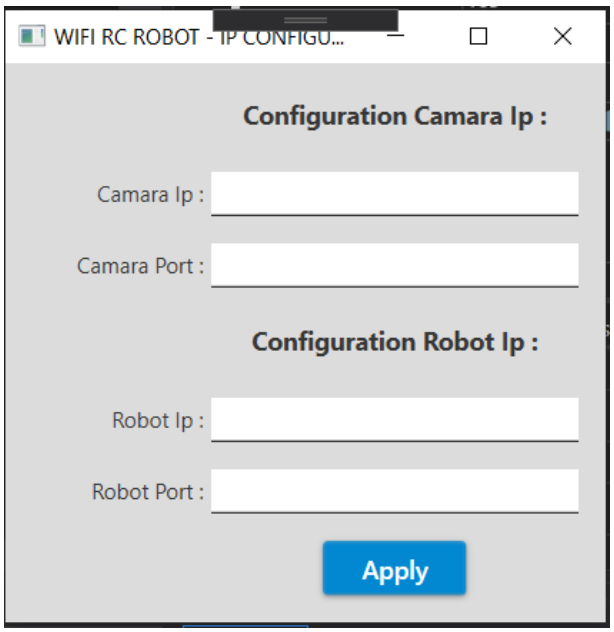


Figure 43: IP Window interface.

A.E Global_Variables.cs

```
public class Global_Variables
{
    //ip variables
    public static string ip_cam;
    public static string ip_rob;
    //port variables
    public static int port_cam;
    public static int port_rob;
}
```

B Arduino Code

```
// UDP WIFI TRANSMISSION BETWEEN ARDUINO AND VISUAL STUDIO
// WIFI ROBOT CONTROLLED USING A XBOX CONTROLLER
// EUDALD SANGENS & GUILLEM CORNELLA

// Include all the libraries
#include <WiFiUdp.h>      // for programming UDP routines
#include <ESP32Servo.h>   // to control the servo motors
#include <WiFi.h>         // to connect to the WIFI module
#include <MapFloat.h>     // to be able to do Maps using float
    variables

// Define the pins of the motor shield module
#define enA 14      // enable the motor A
#define enB 32      // enable the motor B

// Pins that act as a switch, to control the direction of the motor
    :
#define in1 27
#define in2 26
#define in3 25
#define in4 33

// Define the speed of the motors, starting at 0
int motorSpeedA = 0;
int motorSpeedB = 0;

// Defining the servo motors
Servo servo_cam;            // The name of the servo that we are
    using for the camera control
Servo servo_steer;          // The name of the servo that we are
    using for robot steering

// Define the initial position degree of the servos
int initial_pos_cam = 90;   // The initial position of the camera
    will centered at 90 degrees
int initial_pos_steer = 115; // The initial position of the servo
    steer will be at 115 degrees to have the robot straight
int servo_cam_angle_int_y = 0; // The past value of the variable
    that the servo had before the new loop
int servo_steer_angle_int_y = 0;

// Define the PWM pins of the servos
int servo_cam_pin = 4;
```

```
int servo_steer_pin = 5;

// Define the pins of the servos connected to the relay
int servo_cam_relay = 18;
int servo_steer_relay = 19;

// Define the router
const char* ssid = "EBG3808 2.4";      // The name of your WIFI
   router
const char* password = "ventana2.4";  // The password of your
   router

// Define variables of the incoming packets from the C# application
WiFiUDP udp;                          // Create a udp object
unsigned int udpPort = 2000;          // The port to listen to incoming
    packets
char packetBuffer[50];                // Set up a buffer for incoming
   packets (abans era[50])

// Other definitions
#define LED 2                         // Defining a LED connected to
   the GPIO 2
bool led;                             // Define the variable "led" as a
    bool

// Write the code to initialize the board, create the SETUP
void setup() {

  Serial.begin(115200);      // Sets the data rate in bits per
   second (baud) for serial data transmission.
  delay(500);                // Wait for 500ms

  pinMode(enA, OUTPUT);      // Define the pins of the motor shield
   as outputs
  pinMode(enB, OUTPUT);
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(in3, OUTPUT);
  pinMode(in4, OUTPUT);
  pinMode(LED, OUTPUT);      // Define the LED that will act as the
   robot's lights
  pinMode(servo_cam_pin,OUTPUT);
  pinMode(servo_steer_pin,OUTPUT);
  pinMode(servo_cam_relay,OUTPUT);
  pinMode(servo_steer_relay,OUTPUT);
```

```
  servo_cam.attach(servo_cam_pin);        // Attach the servo to the
    pin GPIO 4, the same as D2 for the ESP8266 module
  servo_steer.attach(servo_steer_pin);    // Attach the servo to the
    pin GPIO 5, the same as D1 for the ESP8266 module
  servo_cam.write(initial_pos_cam);       // Set the initial
   position to 0
  servo_steer.write(initial_pos_steer);   // Set the initial
   position to 45

  digitalWrite(LED, LOW);  // Turn off the Led On Board

  // Wifi configuration and connection:
  WiFi.begin(ssid, password);             // Connect to your WiFi
   router
  Serial.println("");                     // Print a blank space
  Serial.print("Connecting");             // Print "Connecting" to
   the Serial Monitor
  while (WiFi.status() != WL_CONNECTED) // Make the Led Flash while
    connecting to the wifi router
  {
    Serial.print(".");                    // Print a dot "." while
   trying to connect
    digitalWrite(LED, LOW);               // Turn off the LED on
   board
    delay(250);                           // Wait for 250ms
    digitalWrite(LED, HIGH);              // Turn on the LED on board
    delay(250);                           // Wait for 250ms
  }
  // If successfully connected to the wifi router, the IP Address
   and port are displayed in the serial monitor
  Serial.println("");
  Serial.print("Successfully connected to : ");
  Serial.println(ssid);
  Serial.print("NodeMCU IP address : ");
  Serial.println(WiFi.localIP());
  udp.begin(udpPort);                   // Once connection is
   established, you can start listening to incoming packets.
  Serial.print("Local Port : ");
  Serial.println(udpPort);              // Print the number of the port
    that's going to be used
}

// Write the loop code to run the program
void loop()
{
  receive_packet();}
```

```
// Waiting to receive incoming UDP packets
void receive_packet()
{
  int packetSize = udp.parsePacket();
  if (packetSize) {
    IPAddress remoteIp = udp.remoteIP();
    int len = udp.read(packetBuffer, 255);
    if (len > 0) packetBuffer[len] = 0;
    String udp_packet_sent = packetBuffer;          // String that
   contains all the data sent by C#.

    Serial.print("UDP packet sent by PC:");
    Serial.println(udp_packet_sent);          // Print in the serial
   monitor the long string containing all the information

// SERVO CAMERA MOVEMENT
    String val_tilt_camera = udp_packet_sent.substring(0,6);//
   Creating a substring from the packet sent to obtain information
   regarding the tilt value "jx0---"
    val_tilt_camera = val_tilt_camera.substring(3,6);         //
   Creating a substring again to obtain the numeric variables and
   removing the letters "jx0"
    float val_tilt_camera_f = val_tilt_camera.toFloat();      //
   Converting the String into a float variable
    val_tilt_camera_f = val_tilt_camera_f - 0.5;              //
   Substracting 0.5 to the value so we can have 0 in the the center
   , -0.5 the minimum, and 0.5 the maximum
    Serial.print("Value of the camera sent by c#:");
    Serial.println(val_tilt_camera_f);
    float servo_cam_angle = mapFloat(val_tilt_camera_f, -0.5, 0.5,
   170.0, 10.0); // Maping the value obtained, -0.5 in the joystick
    equals 170   and 0.5 equals 10
    int servo_cam_angle_int = (int)servo_cam_angle;           //
   Converting the value in degrees into an <int> variable

      if (servo_cam_angle_int == servo_cam_angle_int_y){
        digitalWrite(servo_cam_relay,LOW);
        servo_cam.detach();
        Serial.println("Servo camera OFF");
        delay(200);
      }
      else if (servo_cam_angle_int != servo_cam_angle_int_y){
        digitalWrite(servo_cam_relay,HIGH);
        servo_cam.attach(servo_cam_pin);
        servo_cam.write(servo_cam_angle_int);                 //
```

```
    Move the servo according to the degrees maped
        Serial.println("Servo camera ON");
        Serial.print("Angle of the camera's servo:");
        Serial.println(servo_cam_angle_int);
        Serial.print("Antique angle of the camera's servo:");
        Serial.println(servo_cam_angle_int_y);
        delay(200);
      }
     servo_cam_angle_int_y=servo_cam_angle_int;

// STEER OF THE ROBOT
     String val_steer = udp_packet_sent.substring(6,12);      //
    Creating a substring from the packet sent to obtain information
    regarding the steer value "jx1---"
     val_steer = val_steer.substring(3,6);                    //
    String sent by the PC to control the robot and go right and left
     float val_steer_f = val_steer.toFloat();                 //
    Converting the String into a float variable
     val_steer_f = val_steer_f - 0.5;                         //
    Substracting 0.5 to the value so we can have 0 in the the center
    , -0.5 the minimum, and 0.5 the maximum
     Serial.println("val_steer_f:");
     Serial.println(val_steer_f);
     float servo_steer_angle = mapFloat(val_steer_f, -0.5, 0.5,
    85.0, 145.0); // Maping the value obtained, -0.5 in the joystick
     equals 85 -left and 0.5 equals 145 -right.
     Serial.println(servo_steer_angle);
     int servo_steer_angle_int = (int)servo_steer_angle;      //
    Converting the value in degrees into an <int> variable

     if (servo_steer_angle_int == servo_steer_angle_int_y){
         digitalWrite(servo_steer_relay,LOW);
         servo_steer.detach();
         delay(200);
     }
     else if (servo_steer_angle_int != servo_steer_angle_int_y){
         digitalWrite(servo_steer_relay,HIGH);
         servo_steer.attach(servo_steer_pin);
         servo_steer.write(servo_steer_angle_int);
     // Move the servo according to the degrees maped
         Serial.println("Servo steer ON");
         Serial.print("Angle of the steer:");
         Serial.println(servo_steer_angle_int);
         Serial.print("Antique angle of the steer servo:");
         Serial.println(servo_steer_angle_int_y);
         delay(200);
```

```
    }
    servo_steer_angle_int_y = servo_steer_angle_int;

// ROBOT FORWARD AND BACKWARDS
    String val_vel = udp_packet_sent.substring(12,18);      //
   Creating a substring from the packet sent to obtain information
   regarding the steer value "jy1---"
    val_vel = val_vel.substring(3,6);                       //
   String sent by the PC to control the robot and go forward and
   backwards
    float val_vel_f = val_vel.toFloat();                    //
   Converting the String into a float variable
    val_vel_f = val_vel_f - 0.5;                            //
   Substracting 0.5 to the value so we can have 0 in the the center
   , -0.5 the minimum, and 0.5 the maximum
    Serial.print("Value of the velocity:");
    Serial.println(val_vel_f);

//LEDS ON - LEDS OFF
    String light = udp_packet_sent.substring(18,22);        //
   Creating a substring from the packet sent to obtain information
   regarding the LED value "LED-"
    if (light=="LED1"){                                     // When
    the substring analyzed is "LED1" then light on the LED
      led = true;
      digitalWrite(LED,HIGH);
    }
    else if (light=="LED2"){                                // When
    the substring analyzed is "LED2" then light off the LED
      led = false;
      digitalWrite(LED, LOW);
    }
    else if (light=="LED0"){                                // When
    the substring analyzed is "LED0", if the LED was on, turn it
   off, and vice versa.
      digitalWrite(LED,LOW);
      if (led==true) digitalWrite(LED,HIGH);
      else digitalWrite(LED,LOW);
    }
// Y-axis used for forward and backward control
    // to go BACKWARDS
    if (val_vel_f < -0.1) {
      // Set Motor A backward
      digitalWrite(in1, HIGH);
      digitalWrite(in2, LOW);
      // Set Motor B backward
```

```
      digitalWrite(in3, HIGH);
      digitalWrite(in4, LOW);
      // Convert the declining Y-axis readings for going backwards,
    -0.1 equals 400 and -0.5 equals 1000. The PWM signal is
   increasing the motor speed.
      float motorSpeedA_f = mapFloat(val_vel_f, -0.1, -0.5, 400.0,
   1000.0);
      float motorSpeedB_f = mapFloat(val_vel_f, -0.1, -0.5, 400.0,
   1000.0);
      motorSpeedA = (int)motorSpeedA_f;   // convert the float
   variable of the speed of the motor A to an int
      motorSpeedB = (int)motorSpeedB_f;   // convert the float
   variable of the speed of the motor B to an int
    }
    // to go FORWARD
    else if (val_vel_f > 0.1) {
      // Set Motor A forward
      digitalWrite(in1, LOW);
      digitalWrite(in2, HIGH);
      // Set Motor B forward
      digitalWrite(in3, LOW);
      digitalWrite(in4, HIGH);
      // Convert the growing Y-axis readings for going forward, 0.1
    equals 400 and 0.5 equals 1000. The PWM signal is increasing
   the motor speed.
      float motorSpeedA_f = mapFloat(val_vel_f, 0.1, 0.5, 400.0,
   1000.0);
      float motorSpeedB_f = mapFloat(val_vel_f, 0.1, 0.5, 400.0,
   1000.0);
      motorSpeedA = (int)motorSpeedA_f;    // convert the float
   variable of the speed of the motor A to an int
      motorSpeedB = (int)motorSpeedB_f;    // convert the float
   variable of the speed of the motor B to an int
    }
    // If joystick stays in middle the motors are not moving (to
   avoid warming the motors)
    else {
      motorSpeedA = 0;
      motorSpeedB = 0;
    }
}
  analogWrite(enA, motorSpeedA); // Send PWM signal to motor A
  analogWrite(enB, motorSpeedB); // Send PWM signal to motor B
}
```

C Electrical Diagram