Design and Development of a Robotic System for License Plate Verification in Parking Lots using a Single Shot Detector and ArduRover

By:

Ascandar Alfort	**ID: G202206200**
Mohammed Alhubail	**ID: G202206280**

Final Project

Submitted to

King Fahd University of Petroleum and Minerals

College of Engineering and physics

Department of Control & Instrumentation Engineering

In partial fulfillment of the requirements for the degree of Professional Master in

Robotics and Autonomous Intelligent Systems

Supervised By:
DR. Sami Elferik
Co-Supervised By:
DR. Abdul-Wahid A. AL Saif

August 2023

TABLE OF CONTENTS

Contents

Abstract:

This project presents the design and implementation of an innovative robotic system to enhance parking lot management through automated license plate detection and authorization verification. The system employs advanced image processing techniques and the Single Shot Detector (SSD) algorithm for accurate license plate recognition. Additionally, an autonomous navigation system utilizing ArduoRover ensures efficient and effective movement of the robotic platform within the parking facility.

License plate detection is a crucial component of the proposed system, enabling it to identify and extract license plate information from parked vehicles accurately. The Single Shot Detector algorithm has been chosen for its real-time capabilities and reliable object detection performance. This algorithm enables the robotic system to swiftly analyze images captured by onboard cameras, enabling efficient license plate recognition across varying lighting and environmental conditions.

Authorization verification is a pivotal aspect of parking lot management, ensuring only authorized vehicles are parked. Upon license plate detection, the system cross-references the obtained license plate data with a predefined database of authorized vehicles. This comparison enables the system to verify if a parked vehicle has the necessary parking space permissions.

Integrating an ArduoRover-based autonomous navigation system adds a layer of efficiency to the robotic platform's movement within the parking lot. By utilizing sensor data and predefined maps of the parking facility, the ArduoRover ensures collision-free and optimal routes for the robotic system. This navigation system enhances the overall functionality by reducing human intervention and allowing the automated platform to operate smoothly in dynamic environments.

The outcome of this project is a robust robotic system capable of autonomously detecting license plates, verifying authorization, and efficiently navigating through a parking lot. Combining image processing, license plate detection algorithms, and the ArduoRover navigation system presents a comprehensive solution for enhancing parking lot management and security. The potential benefits include reduced administrative overhead, improved vehicle throughput, and increased safety by preventing unauthorized parking.

In conclusion, the developed robotic system showcases a promising advancement in parking lot management technology, leveraging image processing and autonomous navigation to create an intelligent solution for license plate detection and authorization verification. The project's success underscores its potential applicability in various commercial and public settings, contributing to streamlined parking operations and improved security.

1. Introduction:

In the ever-evolving landscape of urbanization and technological advancements, the management of parking facilities has become an increasingly intricate challenge. The growing number of vehicles on roads has escalated parking-related issues, ranging from unauthorized parking to inefficient space utilization. Addressing these challenges requires innovative solutions that enhance the efficiency of parking lot management and improve security and convenience for both administrators and vehicle owners.

This project aims to develop a cutting-edge robotic system tailored to address the complexities of modern parking lot management. The primary focus of this endeavor is twofold: first, the accurate detection of license plates affixed to parked vehicles, and second, the seamless verification of whether a parked car is authorized to occupy a designated parking space. Achieving this involves integrating sophisticated image processing techniques, advanced license plate detection algorithms utilizing the Single Shot Detector (SSD) approach, and a robust navigation system driven by ArduoRover technology.

The ability to automatically detect and verify license plates is fundamental to this project's objectives. This capability holds immense promise in streamlining the process of identifying and managing parked vehicles within a parking lot. By harnessing the power of image processing, the system can rapidly analyze visual data captured by onboard cameras, extracting essential license plate information with remarkable precision. Incorporating the Single Shot Detector algorithm further enhances the system's real-time processing capabilities, allowing for efficient and accurate license plate recognition across diverse lighting and environmental conditions.

In parallel, verifying authorization status plays an integral role in optimizing parking lot utilization. Only authorized or adequately parked vehicles can lead to congestion, inefficiency, and security concerns. The proposed robotic system alleviates these issues by cross-referencing the detected license plate data with a comprehensive database of authorized vehicles. This verification process ensures that only vehicles with valid permissions are granted access to park in designated spaces, contributing to the overall organization and security of the parking facility.

To facilitate the physical movement of the robotic platform within the parking lot, the project integrates an autonomous navigation system driven by ArduoRover technology. This navigation system harnesses sensor data and preconfigured maps to navigate the robotic platform efficiently and safely through the dynamic and often crowded parking environment. By minimizing human intervention, the ArduoRover-based navigation enhances the system's autonomy, allowing it to adapt to changing conditions and execute tasks with a high degree of reliability.

In summary, this project embarks on a journey to revolutionize parking lot management by harnessing the capabilities of robotic technology. The fusion of image processing, advanced license plate detection algorithms, and the ArduoRover-based navigation system promises to yield a comprehensive solution that optimizes parking space utilization, enhances security, and streamlines administrative operations. The successful development of such a system has the potential to reshape the landscape of parking lot management, contributing to a more efficient and secure urban infrastructure.

2. Literature review

The increasing urbanization and rapid growth in the number of vehicles on roads have led to significant challenges in managing parking facilities efficiently. Consequently, researchers and engineers have been exploring innovative technological solutions to alleviate these challenges. The proposed project aims to develop a robotic system for license plate detection, authorization verification, and autonomous navigation within parking lots, which aligns with this overarching goal. This literature review presents an overview of relevant studies in license plate detection, image processing, single-shot detection algorithms, and autonomous navigation, highlighting their contributions to the project's objectives.

2.1 License Plate Detection and Image Processing:

License plate detection is a fundamental task in vehicle identification and parking management. Researchers have extensively investigated various methodologies for accurate and real-time license plate recognition. Traditional approaches often relied on hand-crafted features and template matching, but these methods must be revised to handle variations in lighting conditions, perspectives, and license plate designs.
Recent advancements in deep learning and image processing have substantially improved license plate detection accuracy. Techniques such as convolutional neural networks (CNNs) have shown exceptional results in detecting and localizing license plates within complex scenes. A notable example is the YOLO (You Only Look Once) algorithm, which demonstrated impressive real-time object detection capabilities, albeit with room for improvement in small object detection and precision.

2.2 Single Shot Detector (SSD) Algorithm:

The Single Shot Detector (SSD) algorithm, known for its balance between speed and accuracy, has gained substantial attention for object detection tasks. SSD overcomes the limitations of other algorithms by employing multiple layers with varying sizes to detect objects of different scales within an image. This adaptability makes SSD well-suited for detecting license plates, which can vary significantly in size due to distance and angle.

2.3 Autonomous Navigation Using ArduoRover:

Autonomous navigation systems enable robotic platforms to traverse complex environments efficiently. The utilization of ArduoRover, an open-source hardware and software platform for rover-type robots, provides an accessible framework for implementing such navigation systems. ArduoRover leverages sensor data, GPS information, and preconfigured maps to ensure safe and effective movement within dynamic environments, making it suitable for parking lot scenarios with varying obstacles and congestion.

Existing studies exploring autonomous navigation using ArduoRover have demonstrated successful implementations in real-world scenarios, including outdoor environments and constrained spaces. These studies emphasize the adaptability and reliability of the platform, suggesting its potential to enhance the efficiency and autonomy of the proposed robotic system within parking lots.

2.4 Integration and Future Directions:

Integrating license plate detection algorithms, particularly the Single Shot Detector, with an ArduoRover-based navigation system represents a novel approach to parking lot management. While individual studies have extensively explored license plate recognition and autonomous navigation separately, there needs to be more literature that holistically combines these technologies to address the specific challenges of parking lots. Thus, the proposed project stands at the forefront of merging these domains, potentially yielding a comprehensive solution with significant implications for enhanced parking facility management, security, and efficiency.

In conclusion, the literature review highlights the advancements in license plate detection, image processing, Single Shot Detection algorithms, and autonomous navigation using ArduoRover technology. The proposed project aims to synergistically combine these technologies to develop a robotic system capable of detecting license plates, verifying authorizations, and autonomously navigating parking lots. Building upon the successes and insights from previous research, this project seeks to contribute to a transformative solution for modern parking lot management.

3. Problem Statement:

The escalating urbanization and rapid growth of vehicle ownership have led to an increasingly intricate challenge in managing parking facilities efficiently and securely. As parking lots become more crowded and diverse in vehicle types, the need for a technologically advanced solution to address unauthorized parking, ensure efficient space utilization, and enhance security has become paramount. Current parking management practices rely on manual checks and administrative interventions, resulting in inefficiencies, human errors, and increased operational costs.

Figure 1

In this context, this project aims to develop a comprehensive robotic system capable of autonomously detecting license plates affixed to parked vehicles and verifying whether the parked car holds authorization to occupy the designated parking space. This system combines cutting-edge technologies, including image processing, license plate detection algorithms utilizing the Single Shot Detector (SSD) approach, and an autonomous navigation system built on the ArduoRover platform.

The primary problem is the need for an integrated and automated solution to swiftly and accurately detect license plates, cross-reference them with authorization databases, and facilitate autonomous navigation within parking lots. Current methods often involve manual checks by parking attendants, which can lead to delays, errors in authorization verification, and increased congestion due to unauthorized parking.

Figure 2

The challenges encompass designing and optimizing a robust license plate detection algorithm that operates effectively across diverse lighting conditions, vehicle orientations, and license plate designs. Additionally, the system must seamlessly integrate license plate data with authorization databases to ensure real-time verification of parking permissions. Furthermore, developing a navigation system using ArduoRover to allow the robotic platform to navigate the dynamic and obstacle-laden environment of parking lots autonomously requires careful consideration.

Overall, the absence of a cohesive solution to detect license plates, verify authorizations, and autonomously navigate parking lots poses a significant operational challenge in modern parking management. This project seeks to bridge this gap by creating an innovative robotic system that addresses the complexities of parking lot management while enhancing security, reducing unauthorized parking, and optimizing space utilization.

4. System Design

4.1 Component Selection

4.1.1 Motor:

This is a low-speed and high-torque 12V DC motor with a gearbox. DC motors are the most common type of motors used in robotics. DC motors have various shapes and sizes. They are categorized into 9 types: permanent magnet iron core, permanent magnet ironless rotor, permanent magnet brushless, wound field series connected, wound field shunt connected, wound field compound connected, variable reluctance stepper, permanent magnet stepper, and hybrid stepper motors.

Figure 3

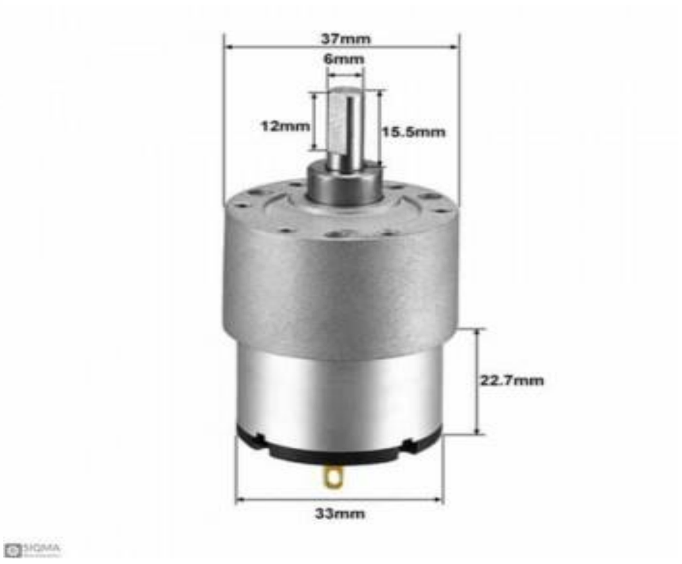


Figure 4

DC motors usually have a high speed and, consequently, low torque. These aspects should be improved for use in most applications, such as robotic applications. Therefore, gearboxes are made for this purpose. By connecting the shaft of a DC motor to a geartrain, the output rotation will be slower, but the output torque that the motor delivers will be significantly higher. DC motors assembled with gear box are called Gearhead DC motors, and they are most suitable for building many hobby robots.

Features:
Model number: ASLONG JGB37-520 12V
Rated Voltage: DC 12V
No load speed: 319rpm
No Load Current: 100mA
Load torque: 0.28kg. Cm
Load Speed: 111 RPM
Load Current: 170mA
Load output: 0.8w
Stall torque: 1.1kg. Cm
Stall Current: 1.2kg. Cm
Ratio: 18.8

Before we moved to this motor, another motor was chosen with an encoder and higher torque to be used for ROS and localization in the map and its description below:

This is a Gear Motor w/Encoder, Model No. GB37Y3530-12V-90EN. It is a powerful 12V motor with a 90:1 metal gearbox and an integrated quadrature encoder that provides a resolution of 16 counts per revolution of the motor shaft, corresponding to 1440 counts per revolution of the gearbox's output shaft.

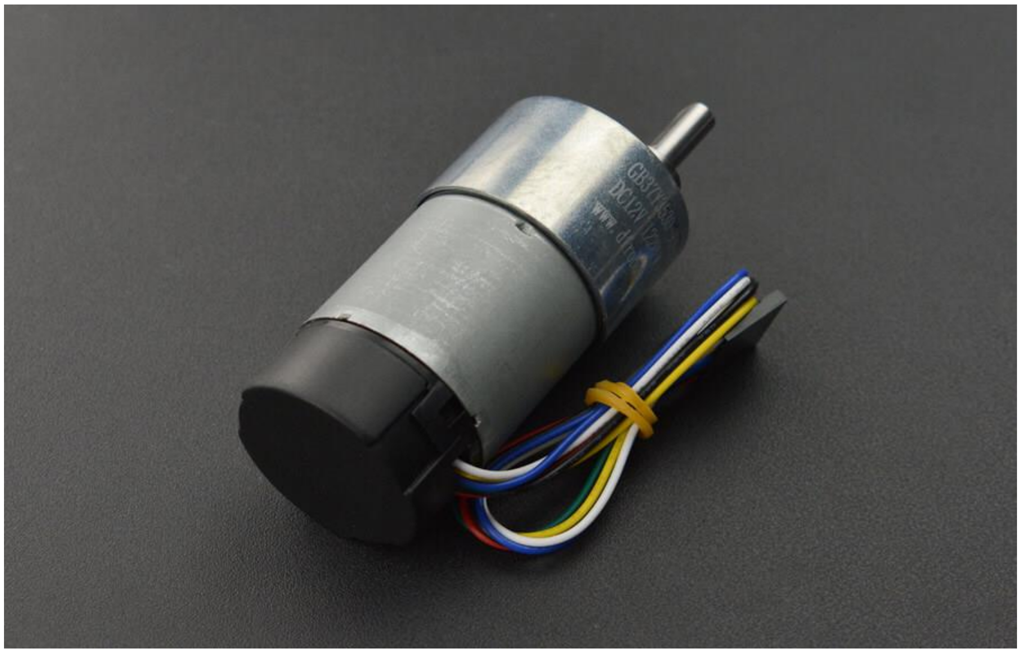


Figure 5

These units have a 0.61" long, 6 mm-diameter D-shaped output shaft. This motor is intended for use at 12V, though the engine can begin rotating at voltages as low as 1V.
The face plate has six mounting holes evenly spaced around the outer edge threaded for M3 screws.

SPECIFICATION
Gear ratio: 90:1
No-load speed: 122+ 10% RPM
No-load current: 350 mA
Start Voltage: 1.0 V
Stall Torque: 38 Kg.com
Stall Current: 7 A
Insulation resistance: 20 M Ω
Encoder Operating Voltage: 5 V
Encoder type: Hall

4.1.2 Main Controller:

In our project, we have two main controllers; the first is capable of controlling and driving the rover, which is Pixhawk 6c, and the second is Jetson Nano.
Pixhawk 6c:

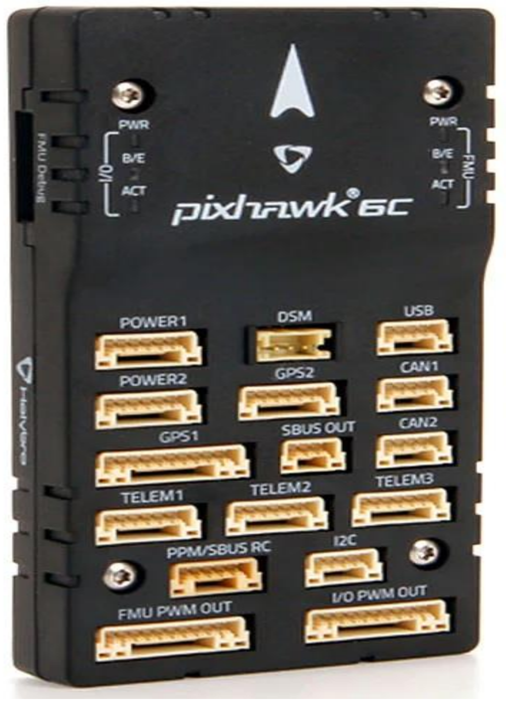


Figure 6

The Pixhawk® 6C is the latest update to the successful family of Pixhawk® flight controllers, based on the Pixhawk® FMUv6C Open Standard and Connector Standard. It comes with PX4 Autopilot® pre-installed. Inside the Pixhawk® 6C, you can find a STMicroelectronics® based

STM32H743, paired with sensor technology from Bosch® & InvenSense®, giving you flexibility and reliability for **controlling any autonomous vehicle**.
The Pixhawk® 6C's H7 microcontroller contains the Arm® Cortex®-M7 core running up to 480 MHz, with 2MB flash memory and 1MB RAM. The FMUv6C open standard includes high-performance, low-noise IMUs on board, designed to be cost-effective while having IMU redundancy. A vibration isolation System to filter out high-frequency vibration and reduce noise to ensure accurate readings, allowing vehicles to reach better overall flight performances.

4.1.3 GPS: attached to Pixhawk® 6C:

The M8N GPS has a UBLOX M8N module, IST8310 compass, a tricolored LED indicator, and a safety switch. There are 3 different connector options for other purposes. This module ships with a baud rate of 38400 5Hz.

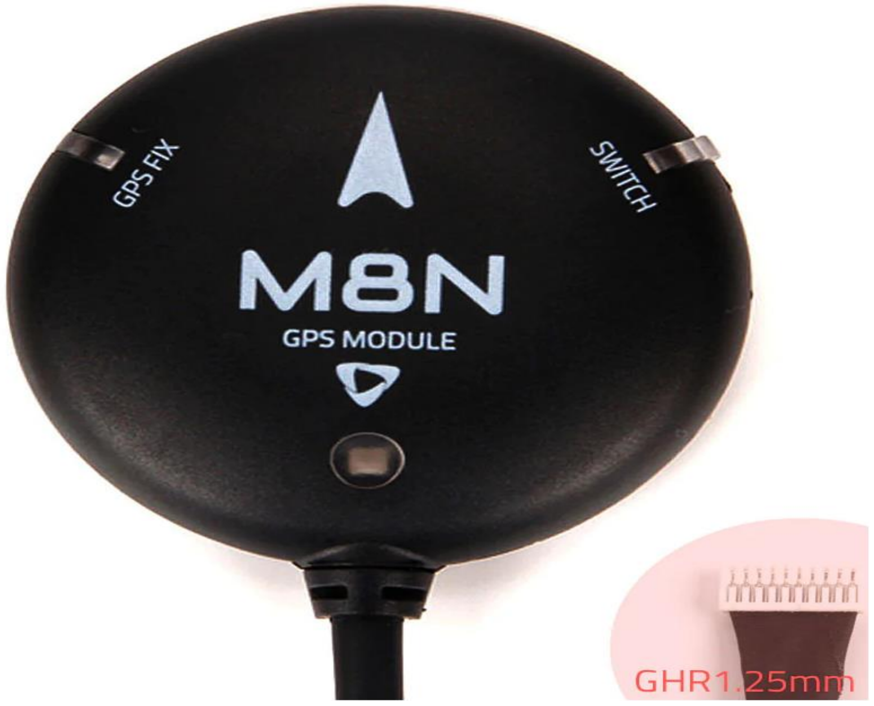


Figure 7

Features and Specifications:
Ublox Neo-M8N module
Industry-leading –167 dBm navigation sensitivity
Cold starts: 26s
LNA MAX2659ELT+
25 x 25 x 4 mm ceramic patch antenna
Rechargeable Farah capacitance
Low noise 3.3V regulator
Current consumption: less than 150mA @ 5V
Fix indicator LEDs
Protective case

Cable Length: 26cm (42cm cable can be purchase here)
Diameter 50mm total size, 32 grams with case.

4.1.4 The second microcontroller is Jetson Nano

: with its robust GPU capabilities, it serves as the system's central. We installed the Single Shot Detector (SSD) model for license plate detection.

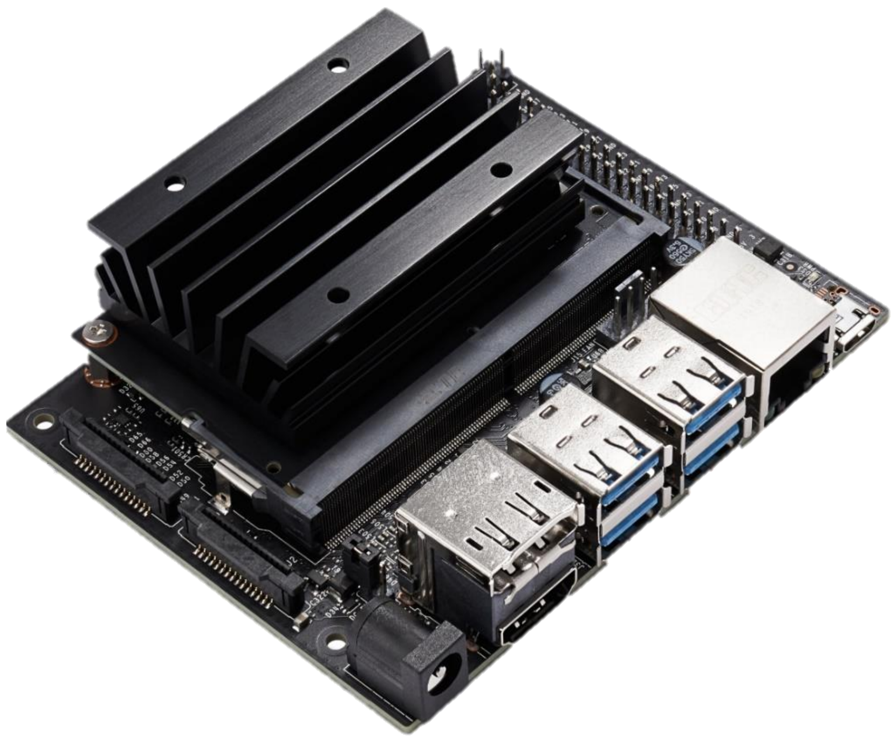
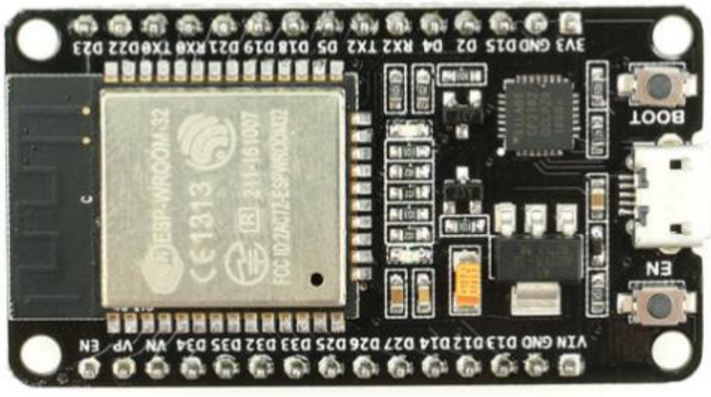


Figure 8

TECHNICAL SPECIFICATIONS:
GPU: 128-core Maxwell
CPU: Quad-core ARM A57 @ 1.43 GHz
Memory: 4 GB 64-bit LPDDR4 25.6 GB/s
Storage: microSD (not included)
Video Encode 4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Video Decode 4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)
Camera: 2x MIPI CSI-2 DPHY lanes
Connectivity: Gigabit Ethernet, M.2 Key E
Display: HDMI and display port
USB 4x USB 3.0, USB 2.0 Micro-B
Others GPIO, I2C, I2S, SPI, UART
Mechanical: 69 mm x 45 mm, 260-pin edge connector

4.1.5 ESP- 32: to connect with the Pixhawch to be used our main microcontroller wirelessly:


3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
BOOT
EN
EN VP VN D34 D35 D32 D33 D25 D26 D27 D14 D12 D13 GND VIN
ESP-WROOM-32
Figure 9

ESP32 is a low-cost, low-power Microcontroller with integrated Wi-Fi and Bluetooth. It is the successor to the ESP8266, which is also a low-cost Wi-Fi microchip, albeit with limited vastly limited functionality.
It is an integrated antenna and RF balun, power amplifier, low-noise amplifiers, filters, and power management module. The entire solution takes up the least amount of printed circuit board area. This board is used with 2.4 GHz dual-mode Wi-Fi and Bluetooth chips by TSMC 40nm low power technology, power and RF properties best, which is safe, reliable, and scale-able to various applications.[00]

4.1.6 Eight Megapixels USB Camera

This is an 8000K pixels driver-free camera with a USB port with up to 3264*2448 resolution. Based on the Sony IMX179 CMOS sensor, it supports autofocus, plug, and play, offering HD photos and video.

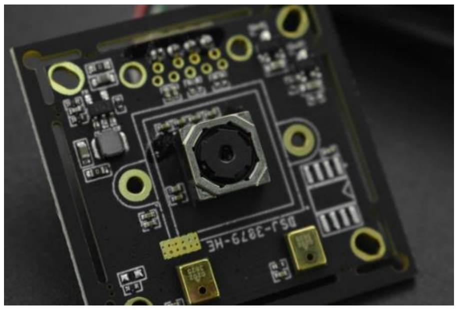


Figure 10

Compatible with multiple systems like Windows, iOS, Android, and Linux, this camera module comes with a digital microphone that captures the sound of 5 meters around the camera when recording video. The design of the USB port also makes it easier to use with your Raspberry Pi and **Jetson Nano**.

SPECIFICATION
Photosensitive Element: CMOS
Sensor: Sony IMX179

Lens Size：1/3.2inch
Pixel Dimension: 1.4µm
Angle: 76°
Image Area: 6.18mm*5.85mm
Max Effective Resolution: 3264(H) X 2448(V) 8 Megapixel
Image Output Format: JPEG
Supported Resolution and Frame Rate:

3264X2448 @ 15fps / 2592X1944@ 15fps
1920x1080 @ 30fps / 1600X1200@ 30fps
1280X720 @ 30fps / 960X540@ 30fps
800X600 @ 30fps   / 640X480@ 30fps

Dynamic Range: 72.5dB
Sensitivity: 0.65V/lux-sec@550nm
Mini illumination: 0.5lux
Shutter Type: Electronic rolling shutter / Frame Exposure
Interface Type: USB2.0 High Speed
Camera Parameter: high-speed photographic apparatus camera, distortionless, size of 1/3.2, Auto-focus, 75 degree
Fill Light Power Supply Port: 4P-2.0 socket
Operating Voltage: DC 5V
Operating Current: 160mA~260mA
Operating Temperature: -10～70℃
Dimension: 38*38mm/1.50*1.50”

4.1.7 Motor driver L298

This L298N Motor Driver Module is a high-power motor driver module for driving DC and Stepper Motors. This module comprises an L298 motor driver IC and a 78M05 5V regulator. L298N Module can control up to 4 or 2 DC motors with directional and speed control.

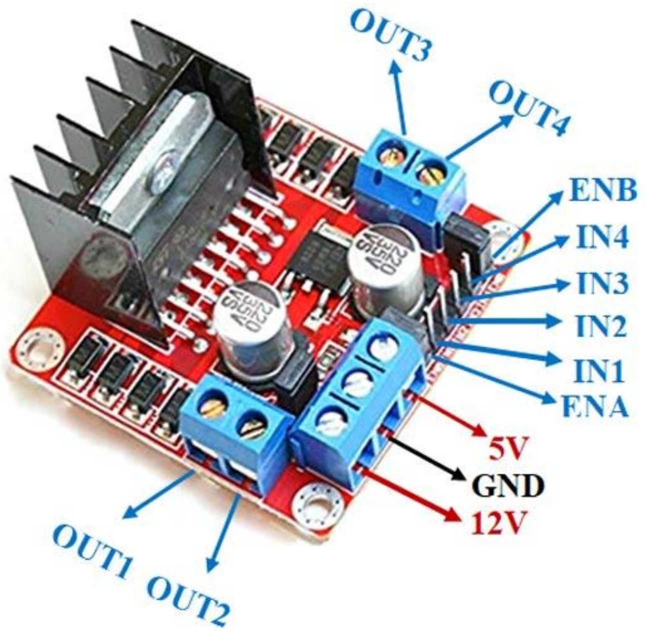


Figure 11

The L298N Motor Driver module comprises an L298 Motor Driver IC, 78M05 Voltage Regulator, resistors, capacitor, Power LED, and 5V jumper in an integrated circuit.

Features & Specifications
Driver Model: L298N 2A
Driver Chip: Double H Bridge L298N
Motor Supply Voltage (Maximum): 46V
Motor Supply Current (Maximum): 2A
Logic Voltage: 5V
Driver Voltage: 5-35V
Driver Current:2A
Logical Current:0-36mA
Maximum Power (W): 25W
Current Sense for each motor
Heatsink for better performance
Power-On LED indicator

4.1.8 RPLIDAR A1M8-R6 - 360 Degree LiDAR Laser:

The system can perform 360-degree scans within a 12-meter range. The produced 2D point cloud data can be used in mapping, localization, and object/environment modeling.

Figure 12

RPLIDAR A1's scanning frequency reached 5.5 hz when sampling 1450 points each round. And it can be configured up to 10 hz maximum. RPLIDAR A1 is a laser triangulation measurement system.
RPLIDAR A1 is based on the laser triangulation ranging principle and uses high-speed vision acquisition and processing hardware developed by Slamtec. The system measures distance data more than 8000 times per second.

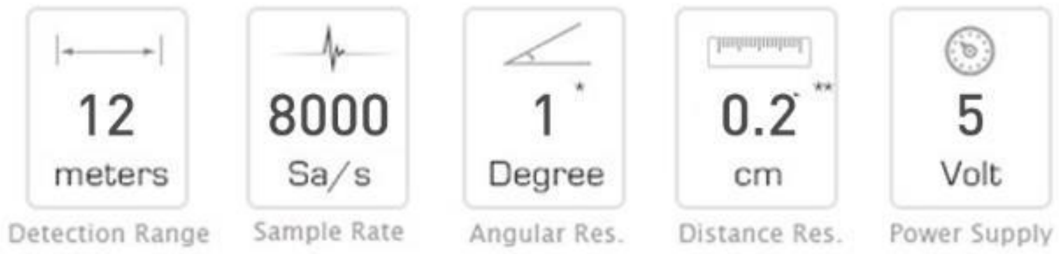


Figure 13

4.1.9 Power bank:

13600mAh high capacity and high power battery. Support fast charging, low self-discharge, and cycles can reach 3000 times.
Also, support 12V vehicle start, and charge for various mobile phone.

Figure 14

Technical specification:
Capacity:1,3600 mAh
Input:12V1A
Output:5V/2A,12V/2A,16V/2A   19V/3.5A
Start Current:200A
Operating Temp:0°C~85°C
Lifetime:1000 cycles

4.1.10 RGB LED

LED strip light can display a single color or multiple colors on one strip. With over 16 million colors and 200 scene modes, you can create more dynamic and gorgeous lighting effects for any occasion.

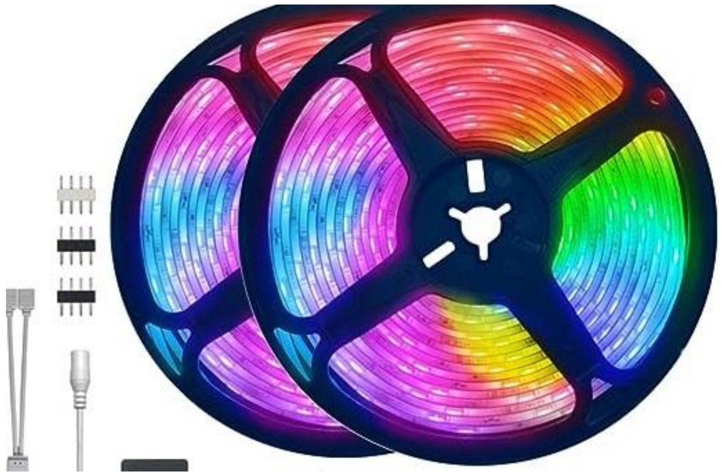


Figure 15

USB Powered: Comes with 5v 2a powered USB cable, which can be plugged into a TV, Computer, Power Bank, etc. It is safe and easy for you to use indoors & outdoors.

5. Electrical connection

First, we start by connecting the 12v out from the power bank to the input of the L298 motor driver. Then, we connect a USB from the power bank to the main microcontroller Pixhawk 6C. In addition, another USB output was connected to power the second microcontroller Jetson Nano and from the Jetson Nano, we powered the ESP-32 and the LED.

5.1 System Circuit

5.1.1 Motor

As mentioned, the Pixhawk 6C is powered by a USB connector. In addition, two PWM signal is used to control these motor through the motor driver.

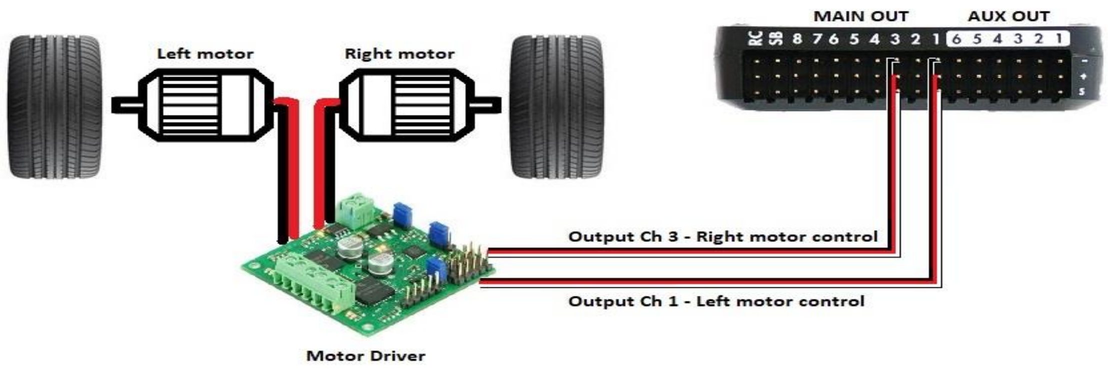


Figure 16

Skid steering will control the direction and forward/reverse motions by varying the speed of two (or more) independent wheels. For these style rovers, the left wheel should be connected to RC Output 1, and the right wheel should be connected to RC Output 3

5.1.2 GPS

Will be connected directly to Pixhawk 6C

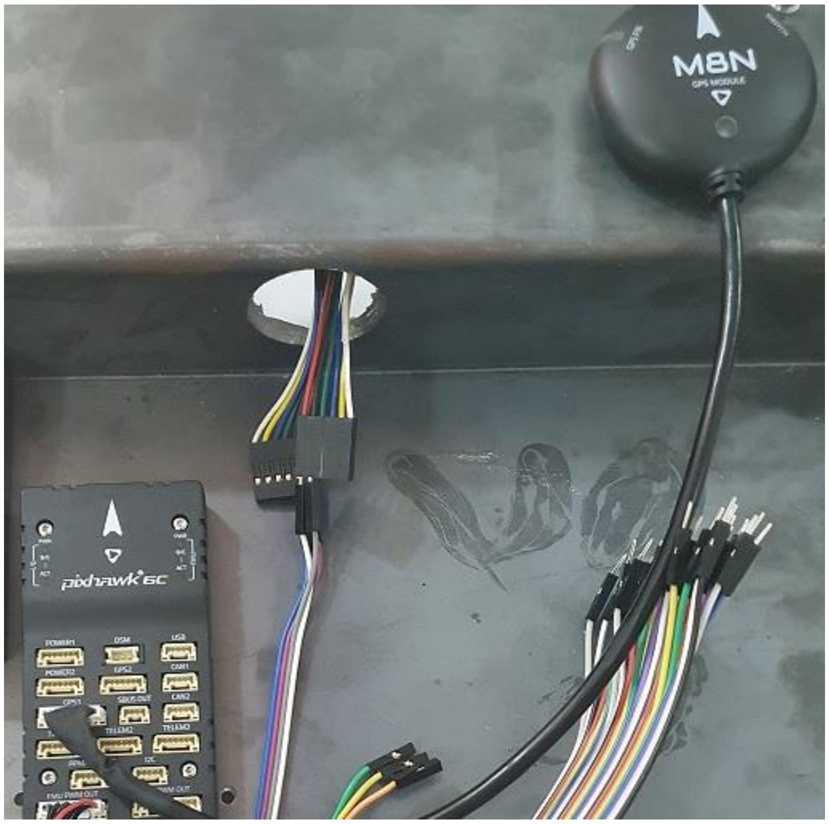


Figure 17

This is the leading GPS in our system and will be used for the path we will create for this autonomous vehicle to follow.

5.1.3 ESP 32

Wiring is straightforward and mostly the same for all devices connected to any serial port (TELEM2)

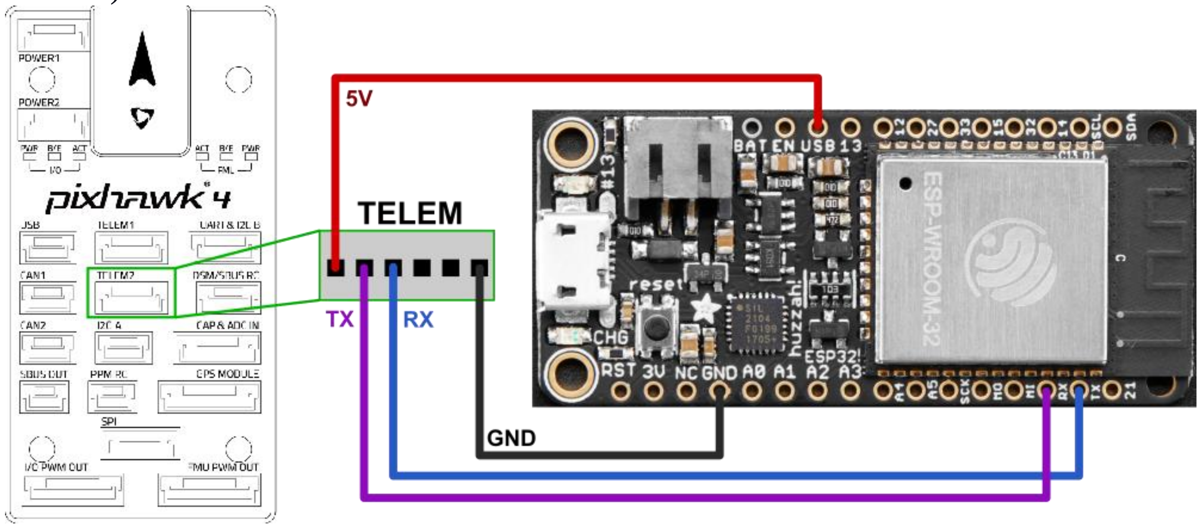


Figure 18

Connect the UART of ESP32 to a UART of your autopilot (e.g., TELEM 1 or TELEM 2 port). Make sure the voltage levels match! Most ESP32 DevKits can only take 3.3V!

TX to RX
RX to TX
GND to GND
Stable 3.3V or 5V power supply to the ESP32 (depending on the available inputs of your DevKit and capabilities of the autopilot)
Set the autopilot port to MAVLINK 1 or 2 protocol.
Some manufacturers of ESP32 DevKits need to correct the labels for the pins on their products. Ensure that your board's PINs are labeled correctly if you encounter issues.

5.1.4 Lidar

The lidar should be mounted horizontally on the top or bottom of the vehicle, with the black cable pointing towards the vehicle's rear. Ensure the sensor's view is not obstructed by any portion of the vehicle, including the GPS mast, vehicle legs, etc.

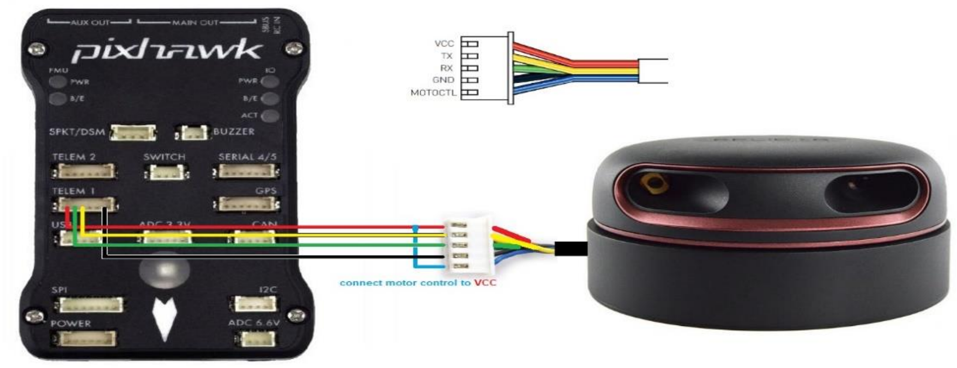


Figure 19

The lidar can be connected to the autopilot's serial input, as shown above. A Pixhawk/Pixhawk2 Telem1 (aka Serial1) should be used because it is more capable of providing the required 1.5A.

5.1.5 Jetson Nano

It is connected directly through UBS for powering with the case for its coaling purpose:

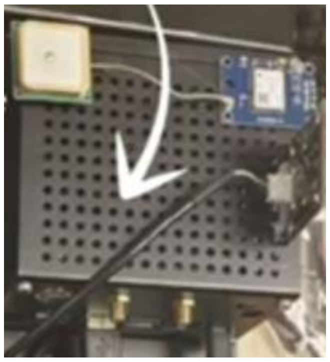


Figure 20

5.1.6 L298 Moor driver

As we mentioned before, the primary power source + the driving signal. As you can see below, the full connection

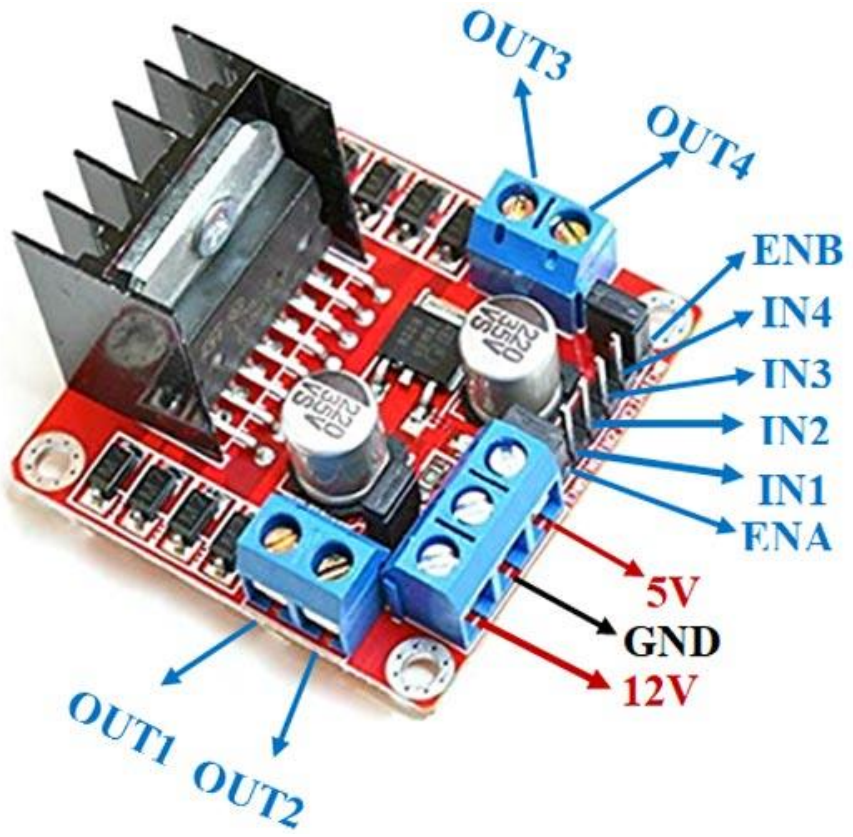


Figure 21

6. Structure Design

The first idea of the design was to do a 3D printing vehicle, and we already made that with all three stages of design, printing, and assembly. However, we changed our design for different reasons, such as size and the ability to handle high temperatures.

6.1 Old Design:

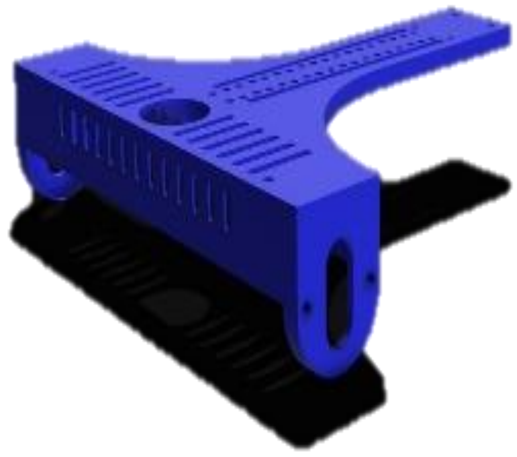


Figure 22

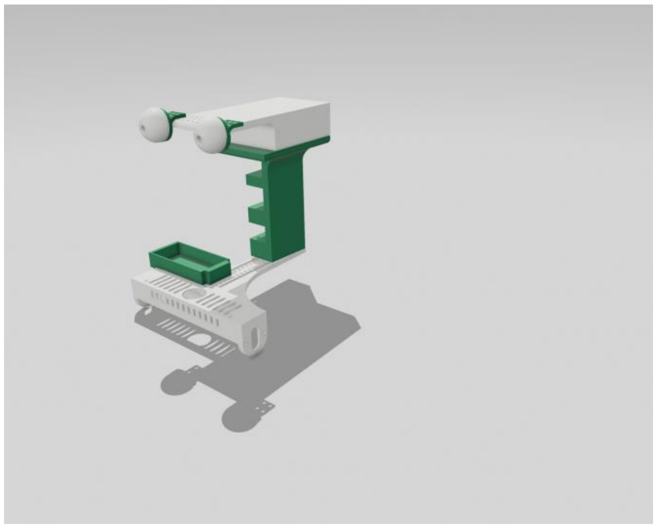


Figure 23

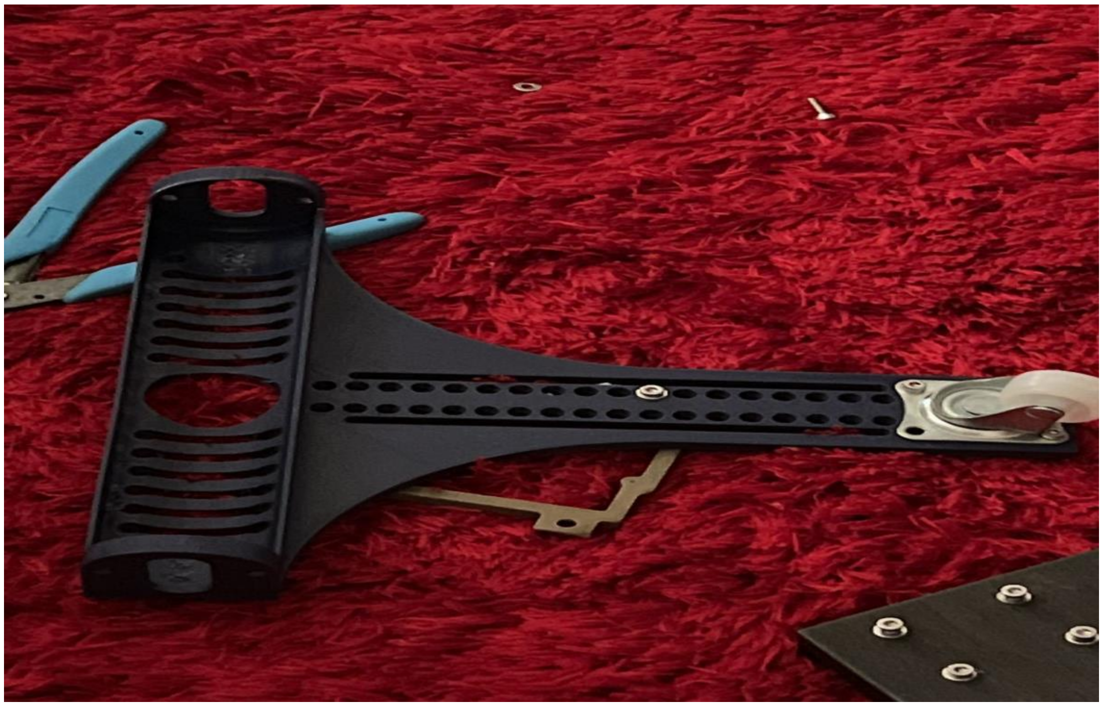


Figure 24

6.2 New Design:

The same old one, but we used a metal plate 40 cm in height and 25 cm in width this time. Also, two motors to drive the vehicle and two only following:

Design:

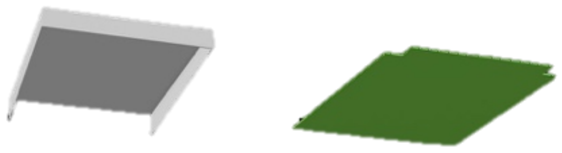


Figure 25

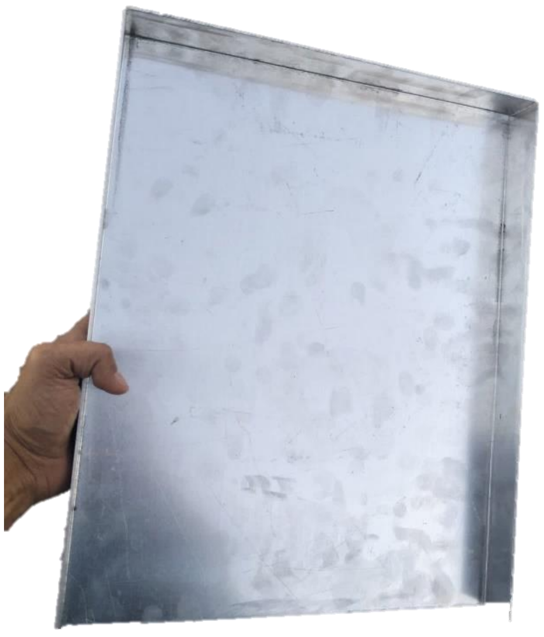


Figure 26

6.3 Assembly:

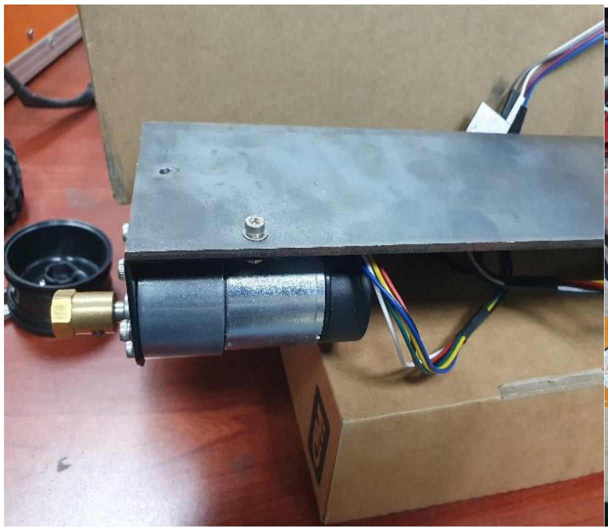


Figure 27

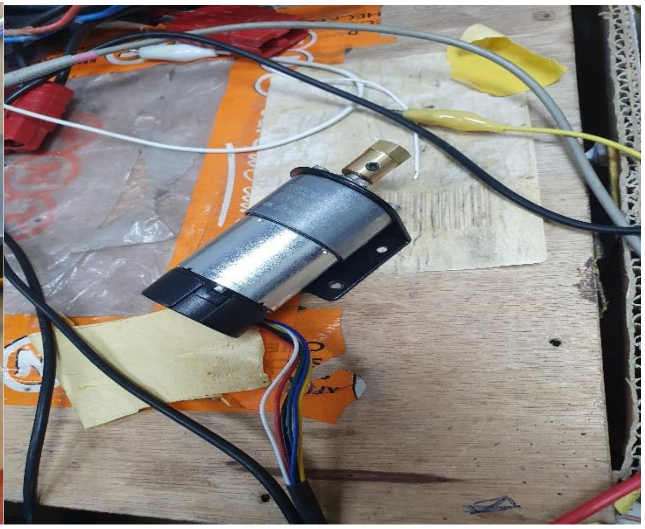


Figure 28

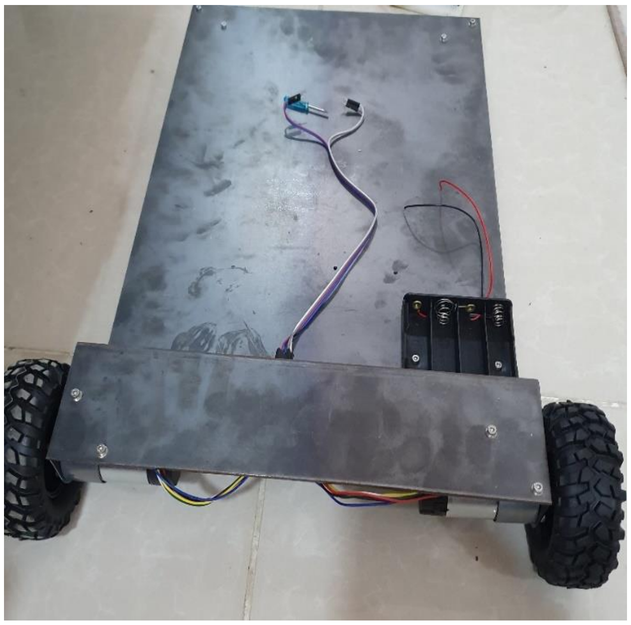


Figure 29

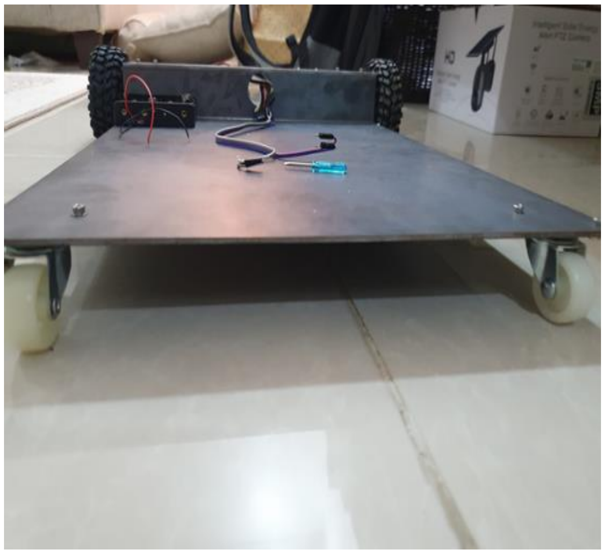


Figure 30

7. Programming

7.1 ROS Programming

1. **Create a URDF (Unified Robot Description Format) file for your robot:** In this step, we designed a URDF file to represent the robot's physical configuration. The URDF format provides a standard way to describe the robot's kinematics, dynamics, and visual properties. By defining joints, links, and sensors in the URDF, we ensured that our robot's model was accurately represented in the simulation and visualization environments

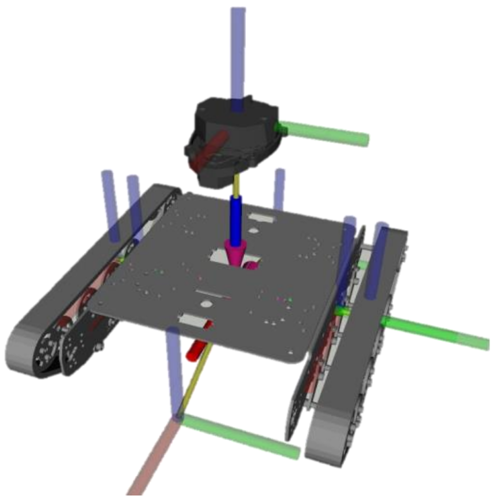


Figure 31

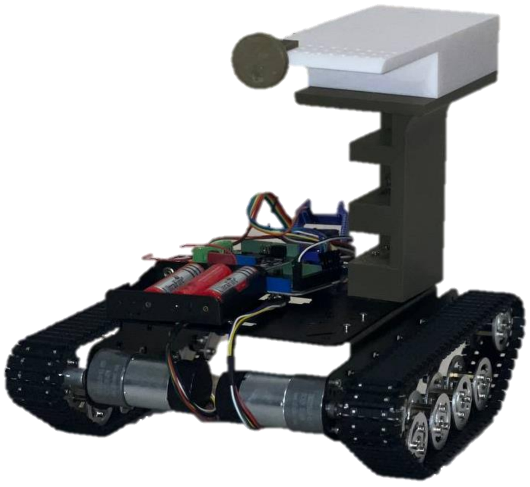
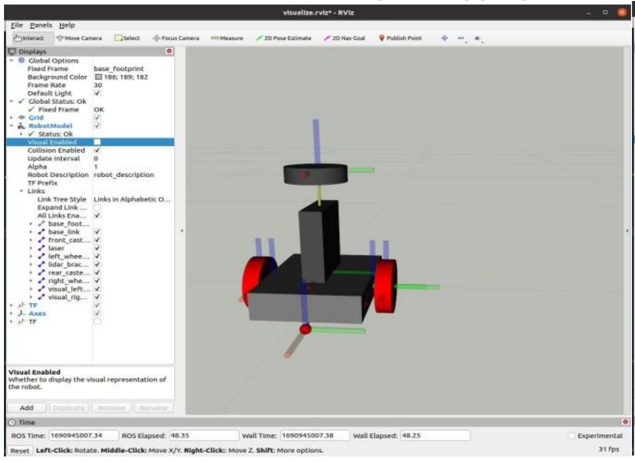
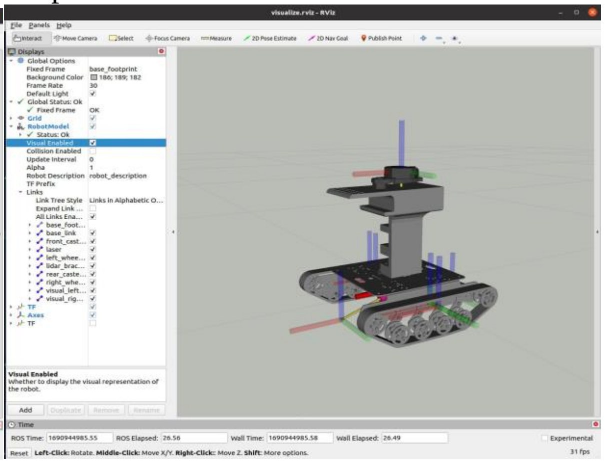


Figure 32

2. **Launch Gazebo and RViz:** Gazebo is a powerful robot simulation tool that allows us to test our robot in various virtual environments without needing physical hardware. After setting up the robot's URDF, we launched Gazebo to visualize and interact with the robot in a simulated world. Alongside Gazebo, we used RViz, a 3D visualization tool for ROS. RViz provides a comprehensive view of the robot, its sensor data, and its interactions with the environment, aiding debugging and development



```
Global Options
  Fixed Frame            base_footprint
  Background Color       186; 189; 182
  Frame Rate             30
  Default Light          ✓
Global Status: Ok
  Fixed Frame            OK
Grid                     ✓
RobotModel               ✓
  Status: Ok
  Visual Enabled         ☐
  Collision Enabled      ✓
  Update Interval        0
  Alpha                  1
  Robot Description      robot_description
  TF Prefix
  Links
    Link Tree Style      Links in Alphabetic O...
    Expand Link ...      ☐
    All Links Ena...     ✓
    base_foot...
    base_link            ✓
    front_cast...        ✓
    laser                ✓
    left_whee...         ✓
    lidar_brac...        ✓
    rear_caste...        ✓
    right_whe...         ✓
    visual_left...       ✓
    visual_rig...        ✓
TF                       ✓
Axes                     ✓
TF                       ☐

Visual Enabled
Whether to display the visual representation of the robot.

ROS Time: 1690945007.34   ROS Elapsed: 48.35   Wall Time: 1690945007.38   Wall Elapsed: 48.25   Experimental
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Move Z. Shift: More options.   31 fps
```



```
Global Options
  Fixed Frame            base_footprint
  Background Color       186; 189; 182
  Frame Rate             30
  Default Light          ✓
Global Status: Ok
  Fixed Frame            OK
Grid                     ✓
RobotModel               ✓
  Status: Ok
  Visual Enabled         ✓
  Collision Enabled      ☐
  Update Interval        0
  Alpha                  1
  Robot Description      robot_description
  TF Prefix
  Links
    Link Tree Style      Links in Alphabetic O...
    Expand Link ...      ☐
    All Links Ena...     ✓
    base_foot...
    base_link            ✓
    front_cast...        ✓
    laser                ✓
    left_whee...         ✓
    lidar_brac...        ✓
    rear_caste...        ✓
    right_whe...         ✓
    visual_left...       ✓
    visual_rig...        ✓
TF                       ✓
Axes                     ✓
TF                       ☐

Visual Enabled
Whether to display the visual representation of the robot.

ROS Time: 1690944985.55   ROS Elapsed: 26.56   Wall Time: 1690944985.58   Wall Elapsed: 26.49   Experimental
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Move Z. Shift: More options.   31 fps
```

•

Figure 33 Figure 34

3. Create a launch file to start all necessary nodes: To streamline the process of starting up our robot's simulation and visualization, we created a ROS launch file. This file specifies which nodes to run, sets necessary parameters, and ensures that all robot system components start in the correct order. Using a launch file, we can initiate our robot's entire software stack with a single command, ensuring consistency and efficiency during testing and development.

4. Modify the Arduino sketch to subscribe to the /cmd_vel topic and publish to the /Speed topic:
We modified an Arduino sketch to bridge the gap between our high-level robot control and the low-level hardware operations. This sketch subscribes to the **/cmd_vel** topic, which provides velocity commands for the robot. Based on these commands, the Arduino computes the necessary motor speeds and actuates the robot's motors accordingly. Additionally, the Arduino sketch publishes the actual motor speeds to the **/Speed** topic, allowing us to monitor and verify the robot's performance in real time.

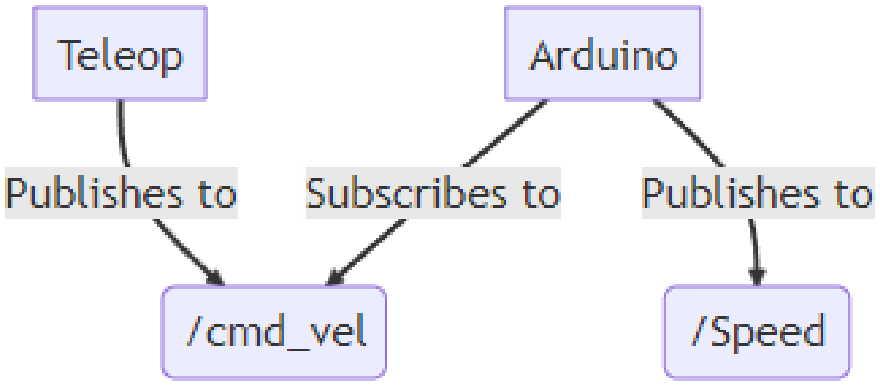


Figure 35

The issue with Topic Subscription from PC:
During the course of our project, we encountered a peculiar issue. While we could subscribe to our robot's topic from the Raspberry Pi (which was operating inside the robot) using SSH, we faced difficulties trying to do the same from our PC. Despite being able to visualize and echo the topic, subscribing to it was unsuccessful.

Upon further investigation, we identified the problem's root cause as our router configuration. Since we were not utilizing a DNS router, port allocation needed to be more consistency. Every time we attempted to look at the topic, the request was redirected to a different port, preventing a successful subscription from our PC.

To address this, we reviewed our network settings and made necessary adjustments to ensure consistent port allocation and communication. By rectifying the router configuration, we could successfully subscribe to the topic from our PC and the Raspberry Pi, ensuring seamless communication and control over our robot.

7.2 Lidar programing:

Flowing parameter should be set in the Pixhawach to set the communication between the lidar and the microcontroller:

SERIAL1_PROTOCOL = “11” (“Lidar360”) if using Serial1
SERIAL1_BAUD = “115” if using Serial1
PRX1_TYPE = “5”
PRX1_ORIENT = “0” if mounted on the top of the vehicle, “1” if mounted upside-down on the bottom of the vehicle.
It is necessary to turn off flow control if using Telem1 (aka Serial1) or Telem2 (aka Serial2)
BRD_SER1_RTSCTS = “0” if using Serial1
Configuring Simple Avoidance for Rover:

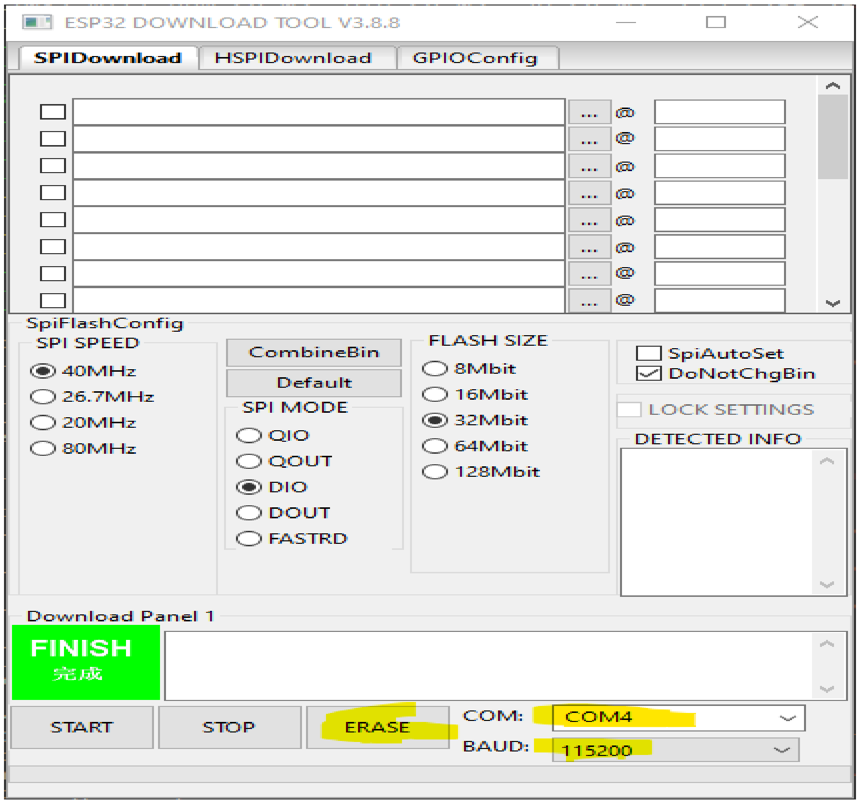
1- Set AVOID_ENABLE = 7 (“All”) to use all sources of barrier information, including “Proximity” sensors.

2- Set PRX1_TYPE = “4” to enable using the first range finder as a “proximity sensor”

Rover attempts to stop the vehicle before it hits objects in all modes except MANUAL.

7.3 ESP-32 Programing:

Downloading and Flashing the Firmware:
https://github.com/DroneBridge/ESP32/releases
Windows only: Use Flash download tools:


ESP32 DOWNLOAD TOOL V3.8.8

SPIDownload | HSPIDownload | GPIOConfig

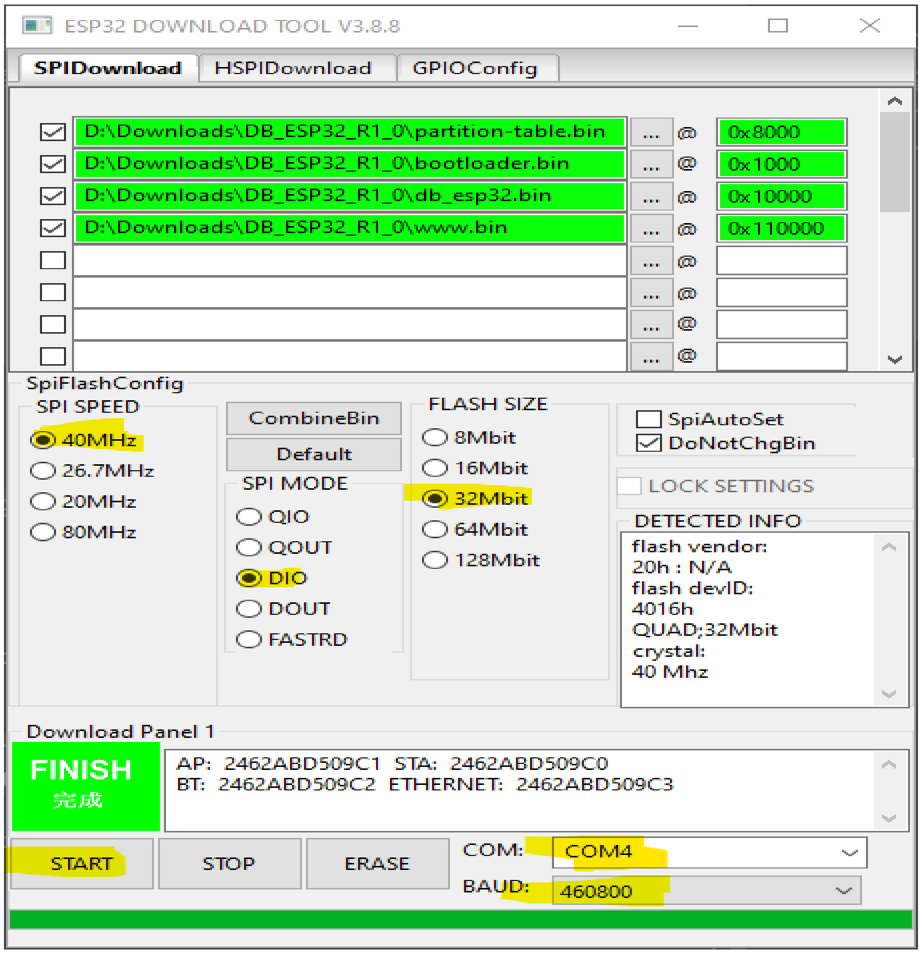
SpiFlashConfig
SPI SPEED: [✓ 40MHz] 26.7MHz 20MHz 80MHz
SPI MODE: QIO QOUT [✓ DIO] DOUT FASTRD
FLASH SIZE: 8Mbit 16Mbit [✓ 32Mbit] 64Mbit 128Mbit
SpiAutoSet ☐
DoNotChgBin ☑
LOCK SETTINGS ☐
DETECTED INFO

Download Panel 1
FINISH 完成

COM: COM4
BAUD: 115200

Figure 36

Erase the flash of the ESP32 before flashing a new release
ESP32 erase flash with flash download tools
Select the firmware, bootloader & partition table and set everything as below
0x8000
0x1000
0x10000
0x110000


ESP32 DOWNLOAD TOOL V3.8.8

SPIDownload | HSPIDownload | GPIOConfig

✓	D:\Downloads\DB_ESP32_R1_0\partition-table.bin	@	0x8000
✓	D:\Downloads\DB_ESP32_R1_0\bootloader.bin	@	0x1000
✓	D:\Downloads\DB_ESP32_R1_0\db_esp32.bin	@	0x10000
✓	D:\Downloads\DB_ESP32_R1_0\www.bin	@	0x110000

SpiFlashConfig
SPI SPEED: [✓ 40MHz] 26.7MHz 20MHz 80MHz
SPI MODE: QIO QOUT [✓ DIO] DOUT FASTRD
FLASH SIZE: 8Mbit 16Mbit [✓ 32Mbit] 64Mbit 128Mbit
SpiAutoSet [✓ DoNotChgBin]
LOCK SETTINGS

DETECTED INFO
flash vendor:
20h : N/A
flash devID:
4016h
QUAD;32Mbit
crystal:
40 Mhz

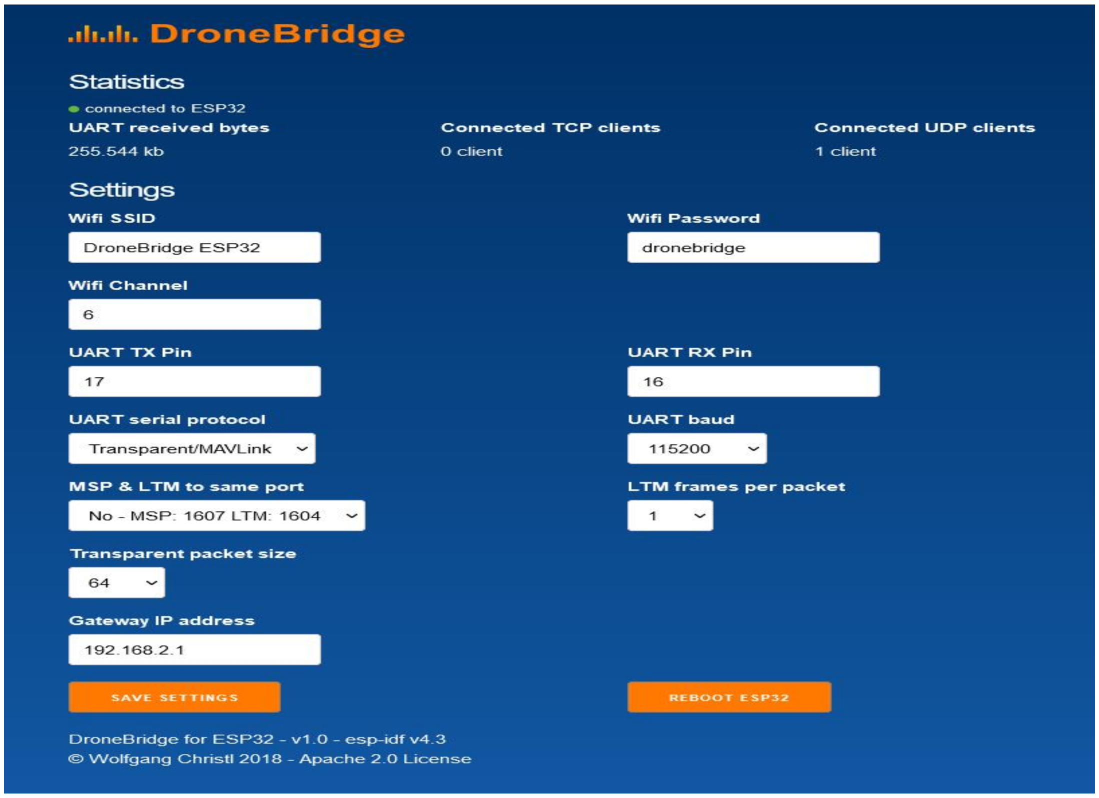
Download Panel 1
FINISH 完成
AP: 2462ABD509C1 STA: 2462ABD509C0
BT: 2462ABD509C2 ETHERNET: 2462ABD509C3

COM: COM4
BAUD: 460800

Figure 37

ESP download tool configuration
Hit Start and power cycle your ESP32 after flashing.
Configuration

1- Connect to the wifi DroneBridge ESP32 with the password drone bridge
2- In your browser, type: dronebridge.local (Chrome: http://dronebridge.local) or 192.168.2.1 into the address bar. You might need to turn off the cellular connection to force the browser to use the wifi connection
3- Configure as you please and hit save


DroneBridge

Statistics

● connected to ESP32

UART received bytes	Connected TCP clients	Connected UDP clients
255.544 kb	0 client	1 client

Settings

Wifi SSID: DroneBridge ESP32	Wifi Password: dronebridge
Wifi Channel: 6	
UART TX Pin: 17	UART RX Pin: 16
UART serial protocol: Transparent/MAVLink	UART baud: 115200
MSP & LTM to same port: No - MSP: 1607 LTM: 1604	LTM frames per packet: 1
Transparent packet size: 64	
Gateway IP address: 192.168.2.1	

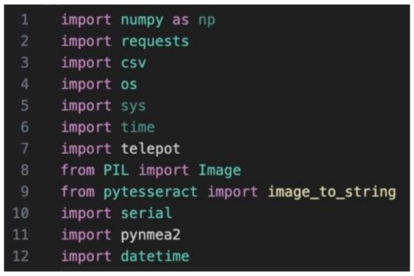
DroneBridge for ESP32 - v1.0 - esp-idf v4.3
© Wolfgang Christl 2018 - Apache 2.0 License

Figure 38

Configuration Options:
Wifi SSID: Up to 31 characters long
Wifi password: Up to 63 characters long
UART baud rate: Same as you configured on your flight controller
GPIO TX PIN Number & GPIO RX PIN Number: The pins you want to use for TX & RX (UART). See the pin out of the manufacturer of your ESP32 device. Flight controller UART must be 3.3V or use an inverter.
UART serial protocol: MultiWii based or MAVLink based - configures the parser
Transparent packet size: Only used with 'serial protocol' set to transparent. Length of UDP packets
LTM frames per packet: Buffer the specified number of packages and send them at once in one

7.4 Jetson Nano programming:

- ***Importing Necessary Libraries***

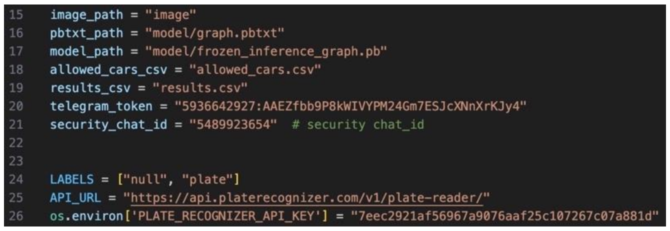


```python
import numpy as np
import requests
import csv
import os
import sys
import time
import telepot
from PIL import Image
from pytesseract import image_to_string
import serial
import pynmea2
import datetime
```

Figure 39

Essential libraries such as numpy, requests, and csv for data handling, os and sys for system-level operations, and telepot for Telegram bot integration are imported. Additionally, pytesseract is used for Optical Character Recognition (OCR), and serial and pynmea2 are for GPS data parsing.

- ***Initialization and Configuration***

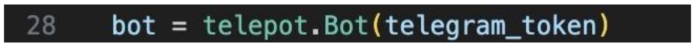


```python
image_path = "image"
pbtxt_path = "model/graph.pbtxt"
model_path = "model/frozen_inference_graph.pb"
allowed_cars_csv = "allowed_cars.csv"
results_csv = "results.csv"
telegram_token = "5936642927:AAEZfbb9P8kWIVYPM24Gm7ESJcXNnXrKJy4"
security_chat_id = "5489923654"  # security chat_id


LABELS = ["null", "plate"]
API_URL = "https://api.platerecognizer.com/v1/plate-reader/"
os.environ['PLATE_RECOGNIZER_API_KEY'] = "7eec2921af56967a9076aaf25c107267c07a881d"
```

Figure 40

Paths for the image, model, and CSV files are defined. The Telegram bot token and chat ID are also initialized. Constants like LABELS and API_URL are set up for later use.

- ***Telegram Bot Setup***

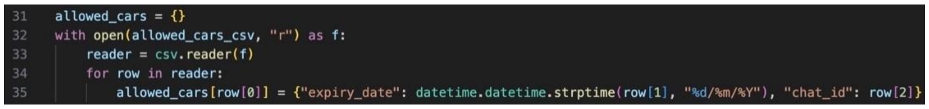


```python
bot = telepot.Bot(telegram_token)
```

Figure 41

The telepot.Bot() function initializes the Telegram bot using the provided token.

- ***Reading Allowed Cars List***

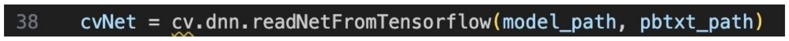


```python
allowed_cars = {}
with open(allowed_cars_csv, "r") as f:
    reader = csv.reader(f)
    for row in reader:
        allowed_cars[row[0]] = {"expiry_date": datetime.datetime.strptime(row[1], "%d/%m/%Y"), "chat_id": row[2]}
```

Figure 42

The code reads a CSV file containing a list of allowed cars and their details, storing them in a dictionary for easy access.

- ***Plate Detector Initialization***

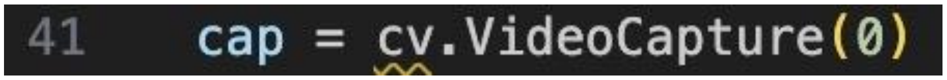


```python
cvNet = cv.dnn.readNetFromTensorflow(model_path, pbtxt_path)
```

Figure 43

The SSD model is loaded using OpenCV's dnn.readNetFromTensorflow() function.

- ***Camera Initialization***

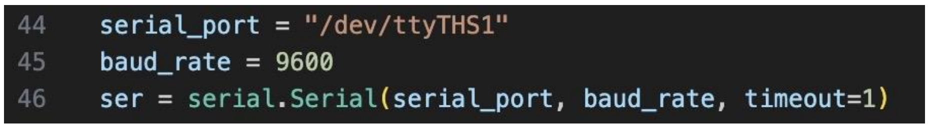


```python
cap = cv.VideoCapture(0)
```

Figure 44

A camera feed is initialized using OpenCV's VideoCapture() function.

- ***GPS Reader Initialization***

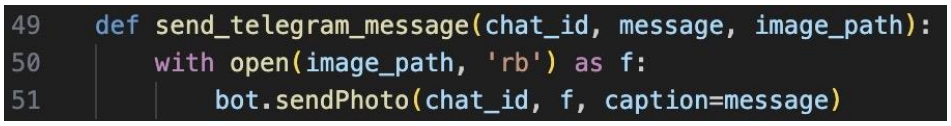


```python
serial_port = "/dev/ttyTHS1"
baud_rate = 9600
ser = serial.Serial(serial_port, baud_rate, timeout=1)
```

Figure 45

The GPS reader is set up using the serial.Serial() function.

- ***send_telegram_message function***

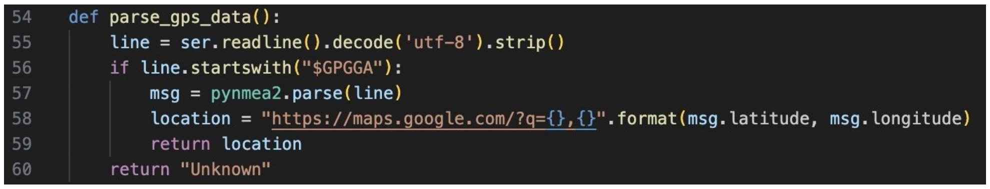


```python
def send_telegram_message(chat_id, message, image_path):
    with open(image_path, 'rb') as f:
        bot.sendPhoto(chat_id, f, caption=message)
```

Figure 46

Sends a photo with a caption to a specified Telegram chat.

- ***parse_gps_data function***

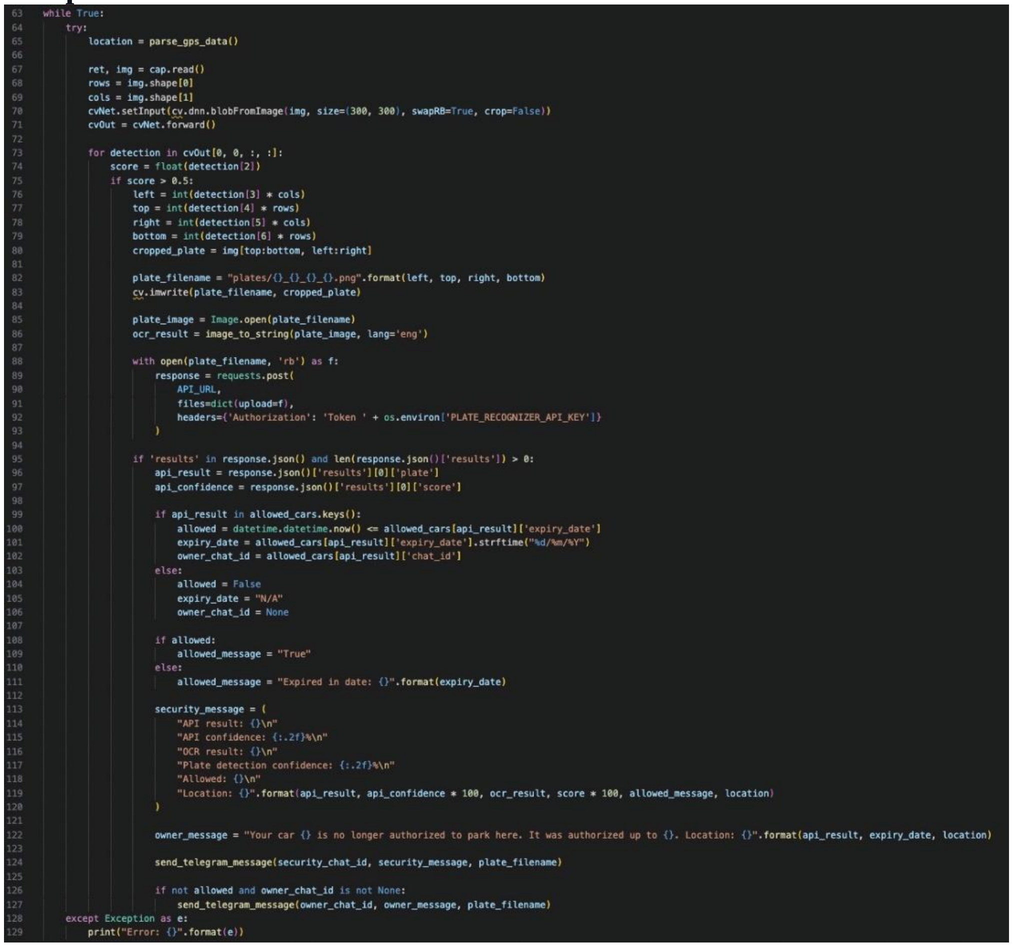


```python
def parse_gps_data():
    line = ser.readline().decode('utf-8').strip()
    if line.startswith("$GPGGA"):
        msg = pynmea2.parse(line)
        location = "https://maps.google.com/?q={},{}".format(msg.latitude, msg.longitude)
        return location
    return "Unknown"
```

Figure 47

Parses GPS data to retrieve the current location.

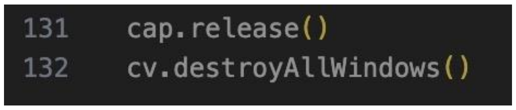
- **Main Loop:**



```python
while True:
    try:
        location = parse_gps_data()

        ret, img = cap.read()
        rows = img.shape[0]
        cols = img.shape[1]
        cvNet.setInput(cv.dnn.blobFromImage(img, size=(300, 300), swapRB=True, crop=False))
        cvOut = cvNet.forward()

        for detection in cvOut[0, 0, :, :]:
            score = float(detection[2])
            if score > 0.5:
                left = int(detection[3] * cols)
                top = int(detection[4] * rows)
                right = int(detection[5] * cols)
                bottom = int(detection[6] * rows)
                cropped_plate = img[top:bottom, left:right]

                plate_filename = "plates/{}_{}_{}_{}.png".format(left, top, right, bottom)
                cv.imwrite(plate_filename, cropped_plate)

                plate_image = Image.open(plate_filename)
                ocr_result = image_to_string(plate_image, lang='eng')

                with open(plate_filename, 'rb') as f:
                    response = requests.post(
                        API_URL,
                        files=dict(upload=f),
                        headers={'Authorization': 'Token ' + os.environ['PLATE_RECOGNIZER_API_KEY']}
                    )

                if 'results' in response.json() and len(response.json()['results']) > 0:
                    api_result = response.json()['results'][0]['plate']
                    api_confidence = response.json()['results'][0]['score']

                    if api_result in allowed_cars.keys():
                        allowed = datetime.datetime.now() <= allowed_cars[api_result]['expiry_date']
                        expiry_date = allowed_cars[api_result]['expiry_date'].strftime("%d/%m/%Y")
                        owner_chat_id = allowed_cars[api_result]['chat_id']
                    else:
                        allowed = False
                        expiry_date = "N/A"
                        owner_chat_id = None

                    if allowed:
                        allowed_message = "True"
                    else:
                        allowed_message = "Expired in date: {}".format(expiry_date)

                    security_message = (
                        "API result: {}\n"
                        "API confidence: {:.2f}%\n"
                        "OCR result: {}\n"
                        "Plate detection confidence: {:.2f}%\n"
                        "Allowed: {}\n"
                        "Location: {}".format(api_result, api_confidence * 100, ocr_result, score * 100, allowed_message, location)
                    )

                    owner_message = "Your car {} is no longer authorized to park here. It was authorized up to {}. Location: {}".format(api_result, expiry_date, location)

                    send_telegram_message(security_chat_id, security_message, plate_filename)

                    if not allowed and owner_chat_id is not None:
                        send_telegram_message(owner_chat_id, owner_message, plate_filename)
    except Exception as e:
        print("Error: {}".format(e))
```

Figure 48

The main loop continuously captures frames from the camera, detects license plates using SSD, and performs OCR on the detected plates. Detected plates are then verified against the API and the allowed cars list. Relevant messages are sent to the security team and car owners via Telegram based on the verification results.



```python
cap.release()
cv.destroyAllWindows()
```

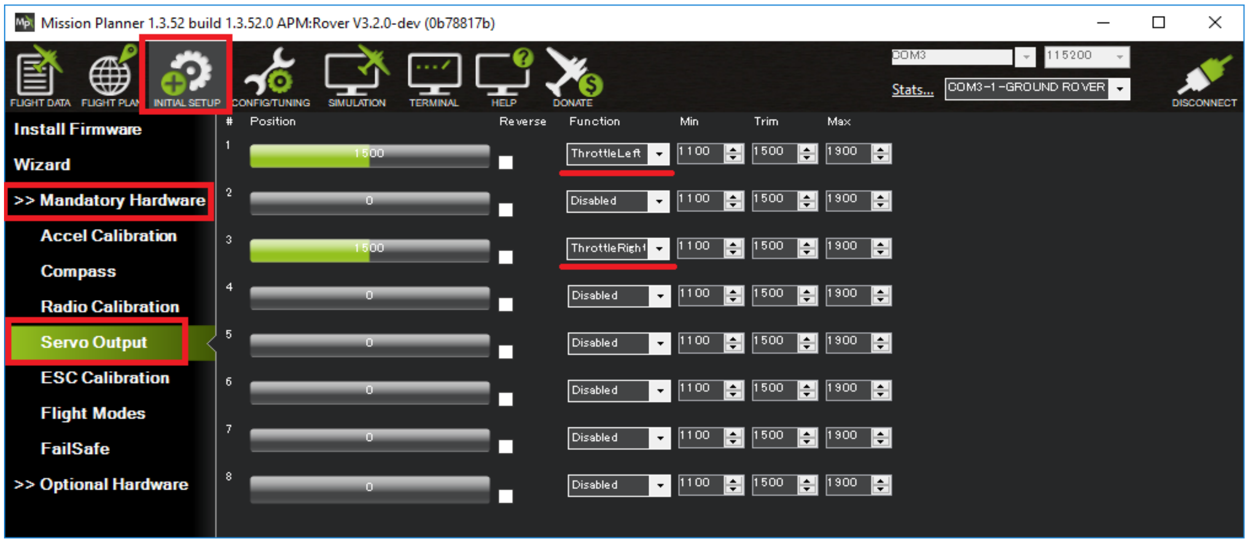
Figure 49

- **Error handling:**
  Any exceptions during the process are caught and printed to the console.
- **Cleanup:**
  The camera feed is released, and any OpenCV windows are destroyed.

7.5 Motor driver programing:

Skid Steering:

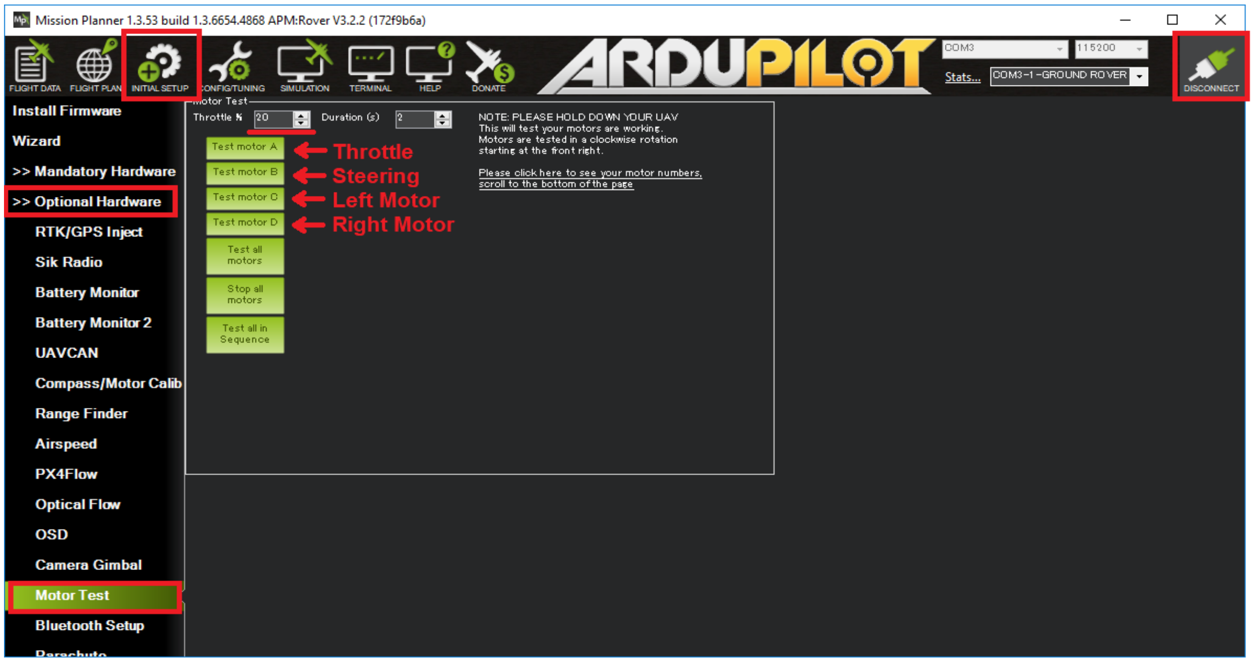
For “Skid steering” vehicles, these parameters values will need to be set:
SERVO1_FUNCTION = 73 (Throttle Left)
SERVO3_FUNCTION = 74 (Throttle Right)


Mission Planner 1.3.52 build 1.3.52.0 APM:Rover V3.2.0-dev (0b78817b)

#	Position	Reverse	Function	Min	Trim	Max
1	1500		ThrottleLeft	1100	1500	1900
2	0		Disabled	1100	1500	1900
3	1500		ThrottleRight	1100	1500	1900
4	0		Disabled	1100	1500	1900
5	0		Disabled	1100	1500	1900
6	0		Disabled	1100	1500	1900
7	0		Disabled	1100	1500	1900
8	0		Disabled	1100	1500	1900

Figure 50

The initial setup to test the motors in the labs is done, and they are working:


Mission Planner 1.3.53 build 1.3.6654.4868 APM:Rover V3.2.2 (172f9b6a)

Throttle % 20   Duration (s) 2

NOTE: PLEASE HOLD DOWN YOUR UAV
This will test your motors are working.
Motors are tested in a clockwise rotation
starting at the front right.

Please click here to see your motor numbers,
scroll to the bottom of the page

Figure 51

8. Results and Discussion

The robot works fine as a prototype. For testing our system, we did more than one experiment test to identify any problem that would occur and to know which components causing this issue.

8.1 First test only for Pixhawk and the GPS

First test we did after uploading the firmware of the rover is connecting the pixhawk microcontroller with GPS module and see if it gives good location indication and test the Mission planer program to see how smooth and accurate it will be

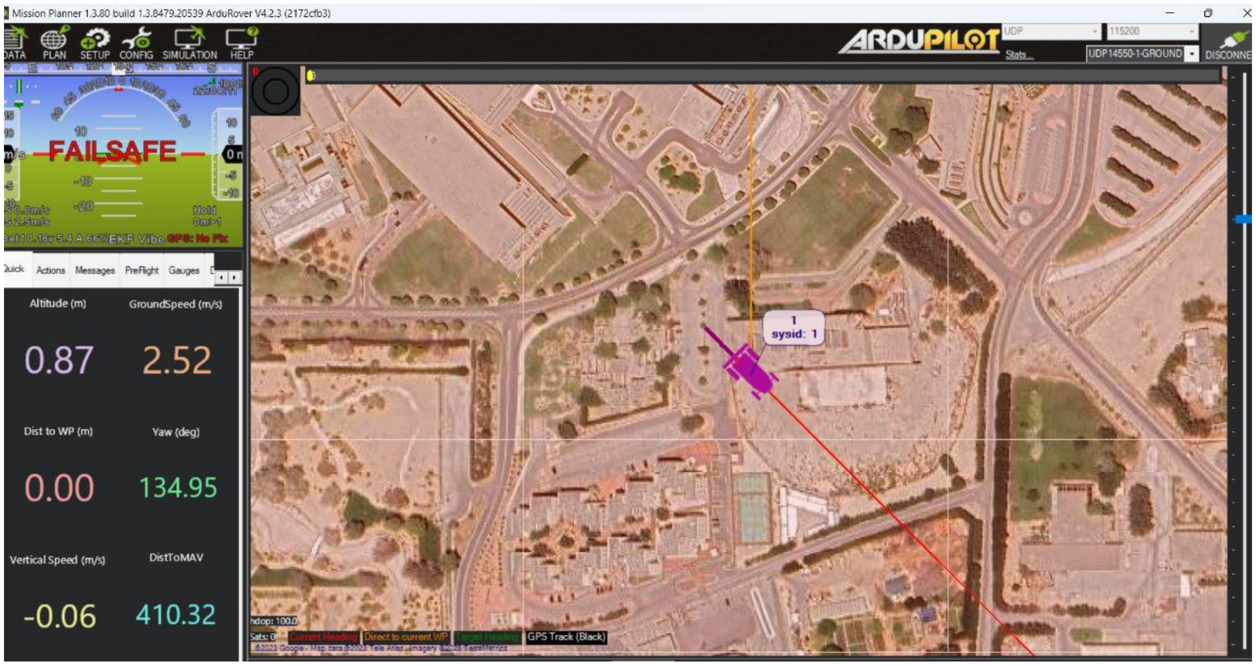


Figure 52

8.2 Second test we add the motor only

As you can see from the picture below, we only connect the motor driver with motor and the microcontroller, but it wasn't succsic test as the motor didn't move at all

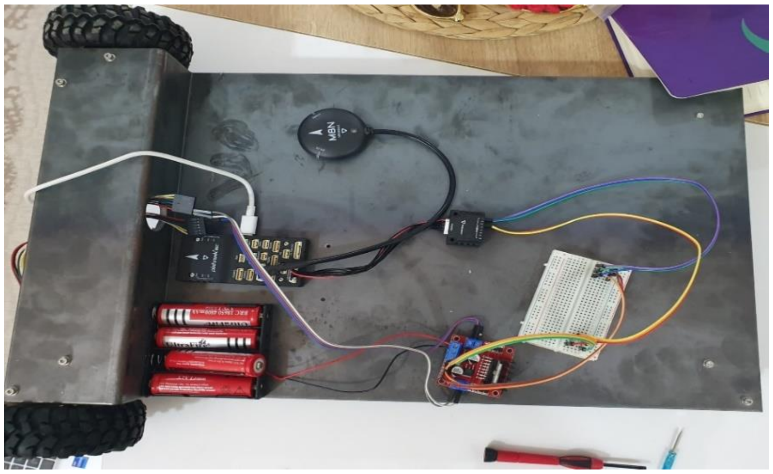


Figure 53

This issue took from us more than a week to solve, as there was no output signal from the microcontroller to drive the motor.
I dismantle all the parts again and take it to the lab to see why there is no output from this microcontroller. The first test was to test it with Digital multimeter, but there wasn't any output at all

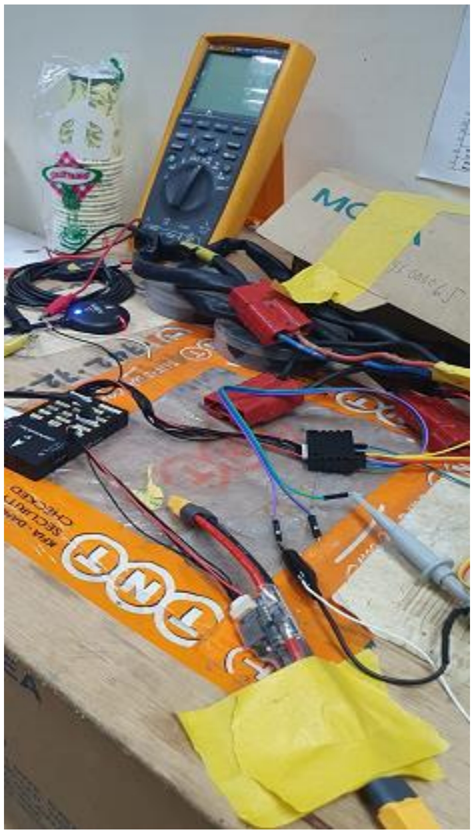


Figure 54

After failing also in this test, I decide to test it with the Oscilloscope as I thought this PWM signal is too much fast and the digital multimeter is not able to capture it, even though this digital multimeter is one of the best.

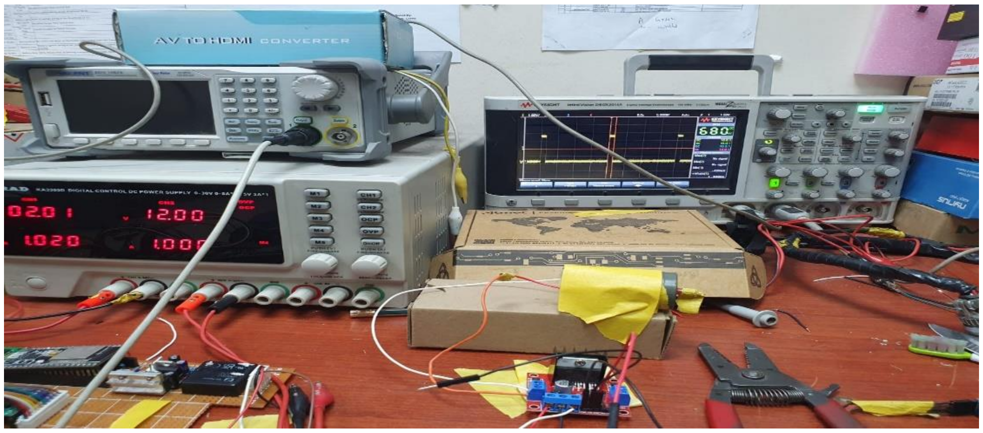


Figure 55

But also, this test wasn't a successful one. After going back to the datasheet and reading more than one I discovered that I am connecting to the wrong pin for the output!

8.3 Third test with the motor driver and ESP-32 for wireless function:

This test was done in small cartoon to be easy to carry it outside and see the life wireless feature is working or not. Also, I connect a DC fan to work as an output from the motor driver to make sure this motor function is working.

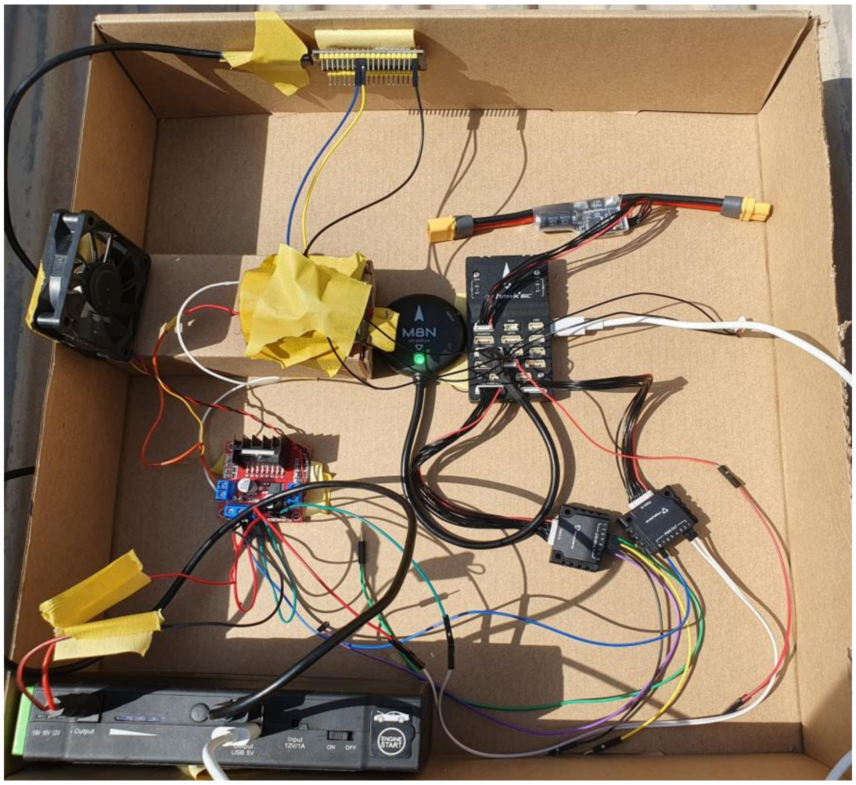


Figure 56

8.4 Testing the Jetson Nano with the camera:

After finish from the programming of this microcontroller, we start the test in the lab by taking picture of a cars and present it in front of the camera to see if it is able to extract the plate number or not

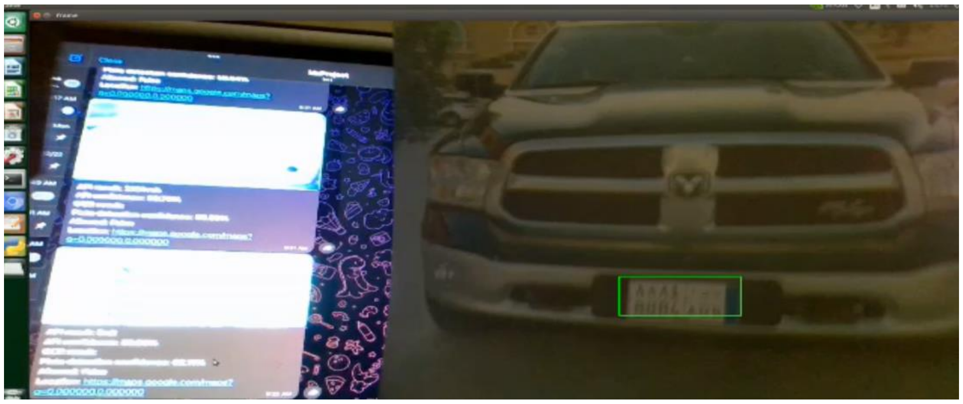


Figure 57

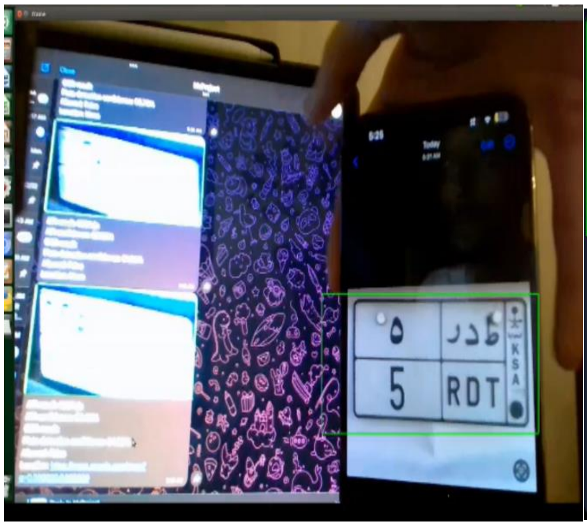


Figure 58

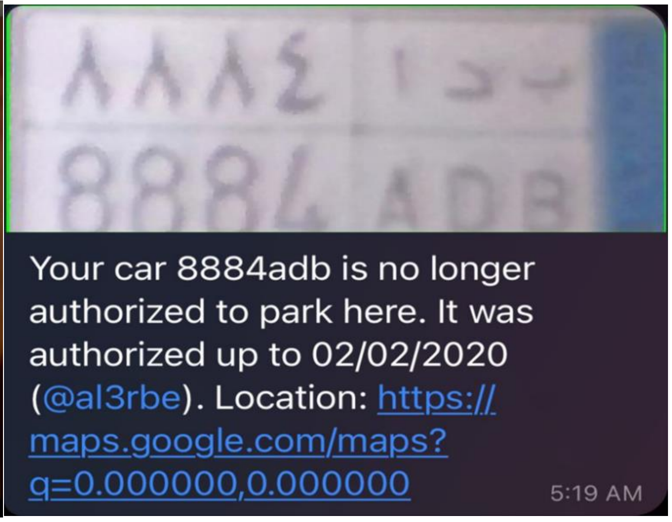

Your car 8884adb is no longer authorized to park here. It was authorized up to 02/02/2020 (@al3rbe). Location: https://maps.google.com/maps?q=0.000000,0.000000 5:19 AM

Figure 59

8.5 Overall movement testing and final assembly

After we test all the parts, and we are sure everything's working we assembled the robot and it look like this:

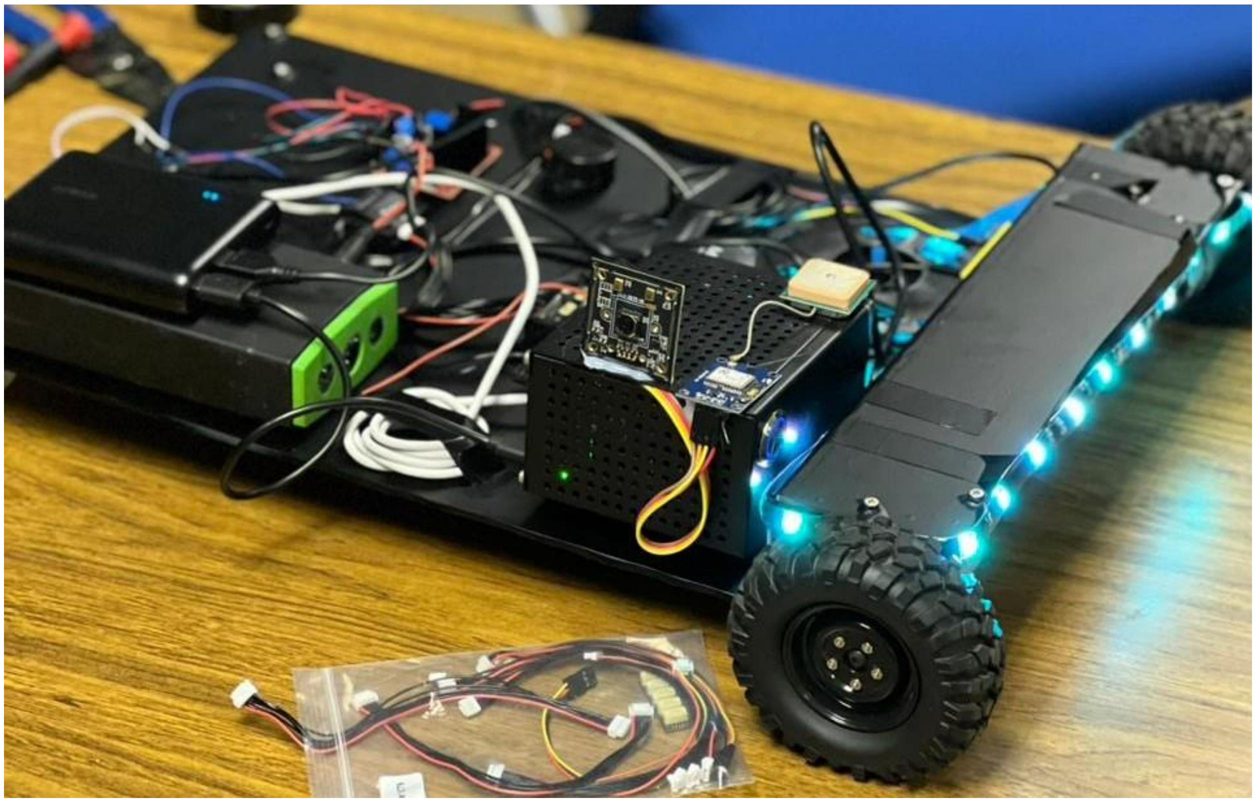


Figure 60

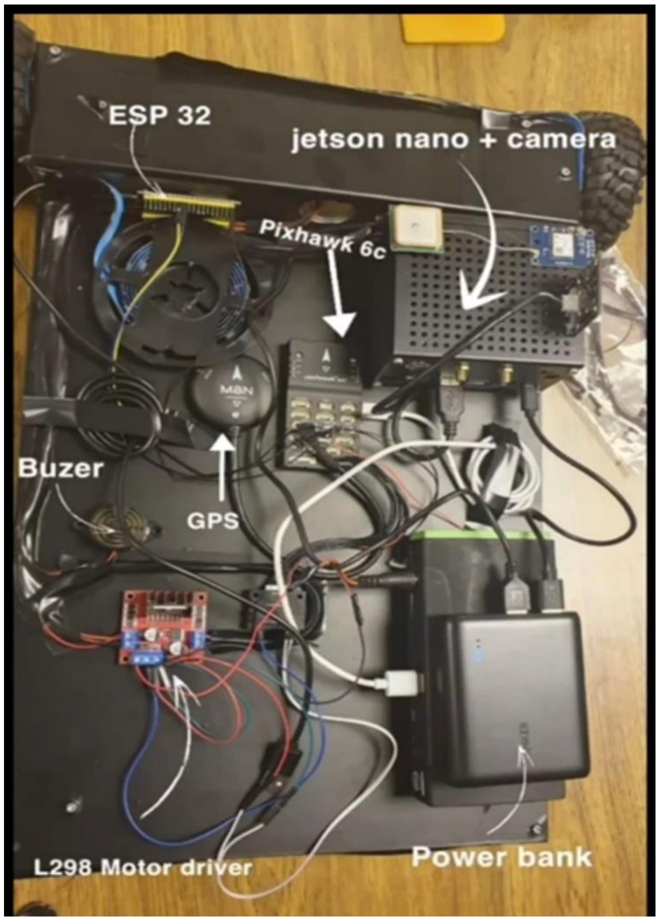


Figure 61

8.6 testing in the parking area

After we finished the assembly, we moved for a real-life test to make sure our robot work in street without any problem

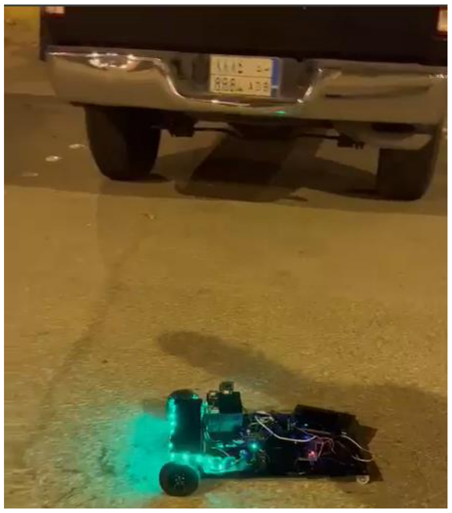


Figure 62

It was a good test and it moved, as it should.

8.7 Final test:

After the prefous tests we are confident about our robot, so we took it back to lab to make it look better and add the LIDAR function. The LIDAR is moving, but it did not work as it should and always give us error not good health LIDAR and due the limited time, we could not fix this issue. The final shape of our robot as following with all the parts install and it is moving without any problem:

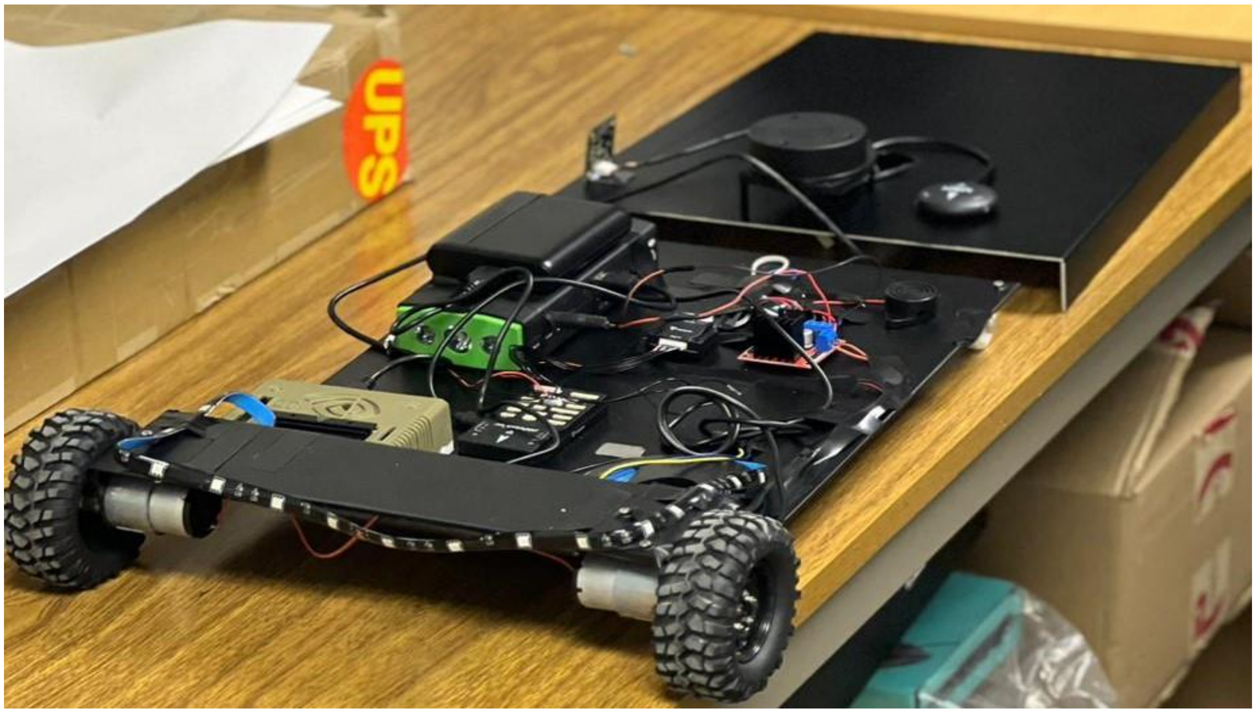


Figure 63

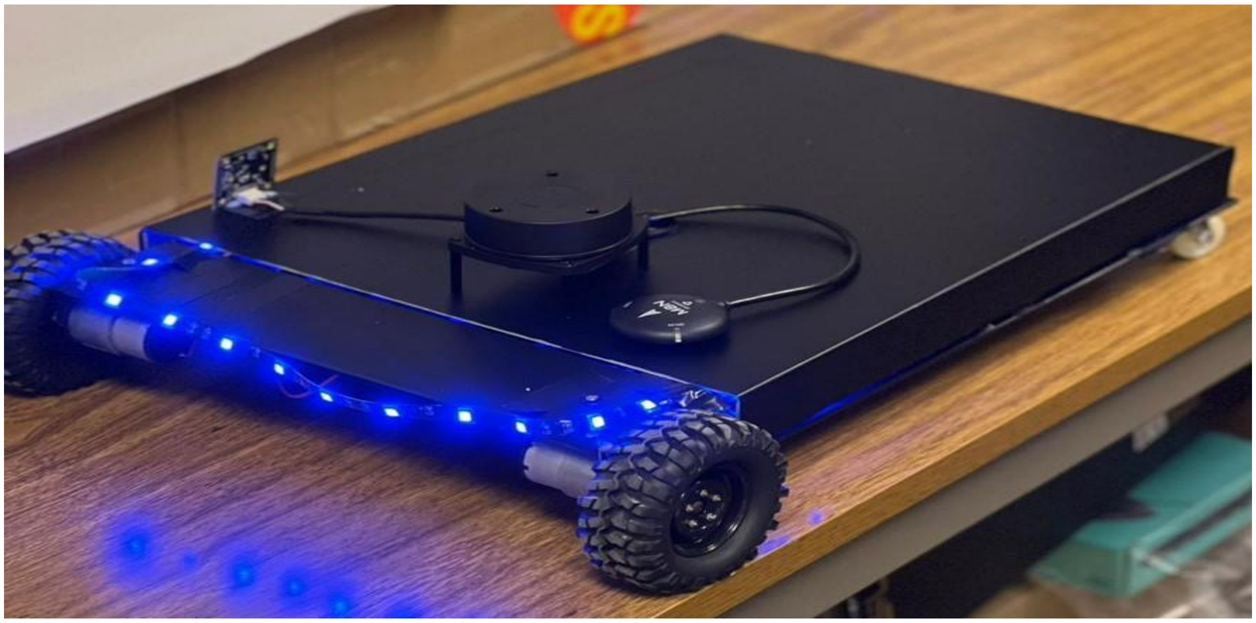


Figure 64

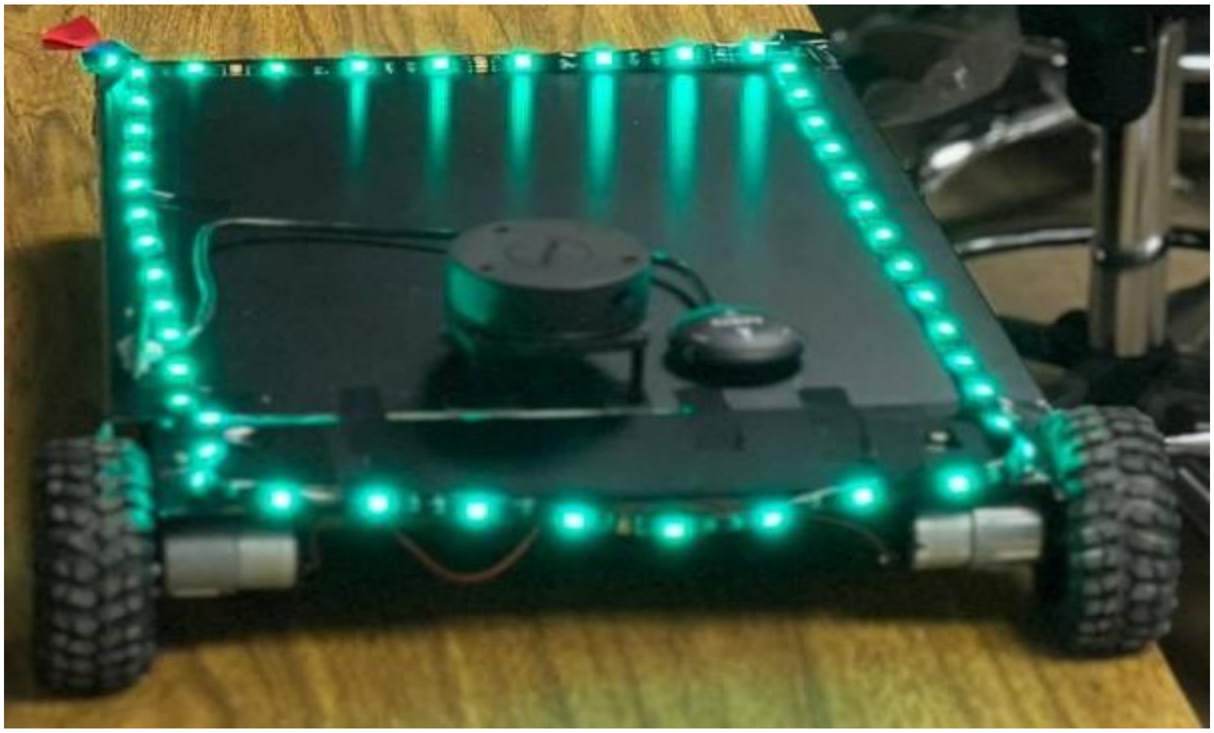


Figure 65

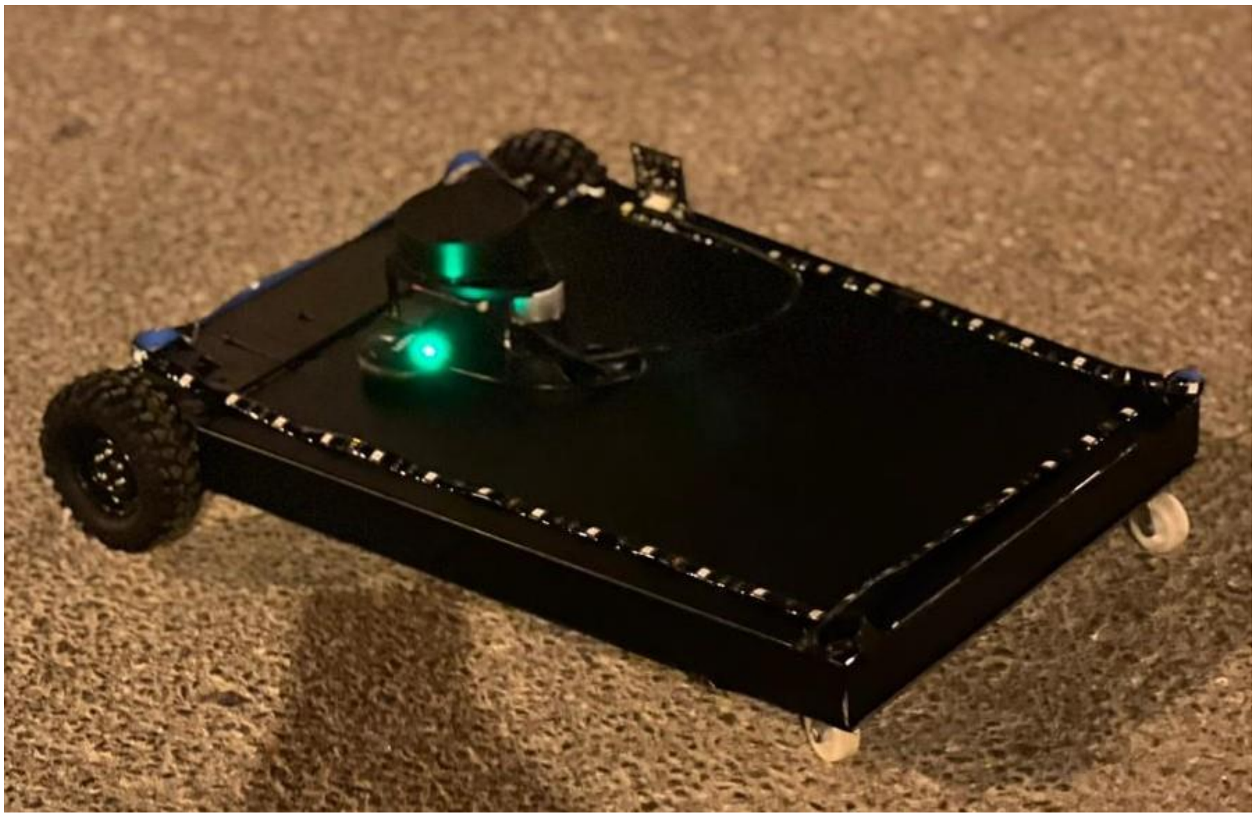


Figure 66

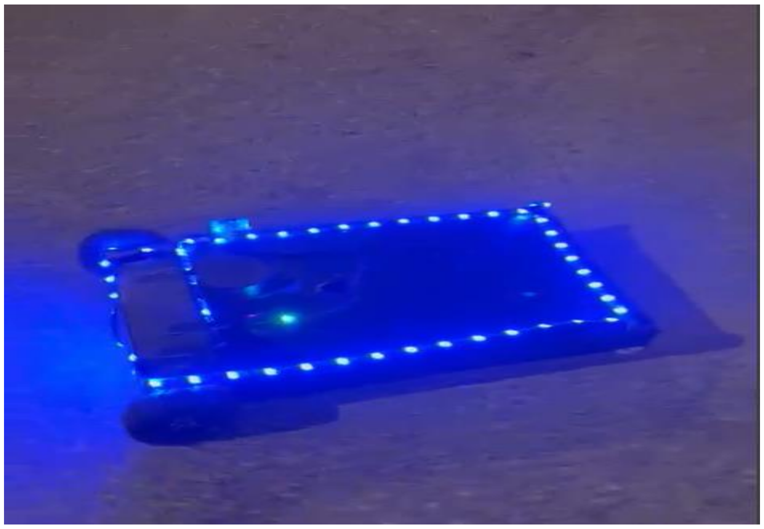


Figure 67

9. Conclusion and Future Directions

In the wake of increasing urbanization and the proliferation of vehicles, efficient parking lot management has emerged as a critical challenge for modern infrastructure. This project aimed to develop a sophisticated robotic system capable of detecting license plates on parked vehicles, verifying their authorization status, and navigating autonomously within parking lots. By integrating image processing, advanced license plate detection algorithms utilizing the Single Shot Detector (SSD) approach, and an autonomous navigation system based on ArduoRover technology, the project sought to revolutionize parking facility management.

This endeavor made significant strides in addressing the multifaceted challenges of parking lot management. The integration of image processing techniques allowed the system to accurately capture and analyze license plate data, overcoming variations in lighting conditions and vehicle orientations. Implementing the SSD algorithm facilitated rapid and precise license plate detection, ensuring real-time processing capabilities vital for managing parking lots with diverse vehicle inflows.

Moreover, the authentication process was streamlined by cross-referencing the extracted license plate data with authorization databases, reducing the potential for unauthorized parking and enhancing overall security. The integration of ArduoRover technology further enhanced the project's efficacy by enabling the robotic platform to autonomously navigate parking lots, reducing congestion and optimizing space utilization.

9.1 Future Directions:

The successful development of this robotic system opens avenues for further advancement in parking lot management and automation. Several directions for future research and development are worth exploring:

Enhanced Vehicle Identification: While the Single Shot Detector algorithm demonstrated impressive performance, ongoing research can focus on refining and customizing the algorithm to handle specific challenges posed by diverse license plate designs and variations in vehicle types.

Real-time Authorization Updates: The system's effectiveness can be augmented by incorporating real-time authorization updates, enabling seamless integration with central databases and instant notification of parking violations or unauthorized entries.

Multi-Sensor Integration: Additional sensors, such as LIDAR or ultrasonic sensors, could enhance the accuracy of the autonomous navigation system, enabling the robotic platform to detect obstacles and navigate more effectively in complex parking environments.

Scalability and Deployment: Future work could explore the system's scalability to accommodate more extensive parking facilities and diverse environments. Additionally, research into deployment strategies considering cost-effectiveness and ease of integration could further its practical application.

Security Measures: As security is paramount in parking lot management, integrating features like facial recognition, surveillance, and anomaly detection could contribute to enhanced security protocols and the prevention of fraudulent activities.

User Interaction and Communication: Developing user-friendly interfaces and communication channels could enable users to interact with the robotic system for inquiries, emergencies, and authorization requests.

Energy Efficiency: Exploring energy-efficient hardware and algorithms could extend the operational duration of the robotic platform, enhancing its efficiency and sustainability.

In conclusion, the developed robotic system showcases the potential of technology to revolutionize parking lot management through license plate detection, authorization verification, and autonomous navigation. As the project concludes, it paves the way for future innovations that can further refine, expand, and tailor this solution to meet the evolving challenges of modern urban infrastructure.

10. Project Challenges and issues faced

1- There was a logistic challenge as we faced a problem with the custom for the Pixhawk as it used as flight controller

2- Limited time after all the parts received to finish assemble, wiring, and programing

3- After we purchase this A1 LIDAR we just discovered it work for indoor environment and it was too late and expensive to replace it with A2 model as it four time the price

4- Hardware Limitations with Raspberry Pi
Issue: Inadequate Computational Power for YOLO.
Detailed Issue: The Raspberry Pi, a popular choice for many DIY projects, is designed for something other than heavy computational tasks. The Raspberry Pi's limited CPU and RAM resources become evident when attempting to run sophisticated object detection algorithms like YOLO. This limitation can lead to significant lags, frame drops, and even system crashes during real-time processing.
Potential Fix: To enhance its processing capabilities, consider upgrading to a more powerful single-board computer or integrating an external GPU with the Raspberry Pi.

5- Camera Quality Concerns:
Issue: Inadequate Image Clarity for Detection.
Detailed Issue: The current camera's sensor might not be sensitive enough to capture high-resolution images under all conditions. Factors like lens quality, sensor size, and pixel density are crucial in image clarity. In challenging conditions, such as low light or high-speed motion, the camera's limitations become more pronounced, leading to blurry or grainy images.
Potential Fix: Upgrade to a camera with higher resolution, better low-light performance, and a wider field of view to capture more details.

6- Need for Advanced Hardware Beyond Jetson Nano:
Issue: Jetson Nano's Limitations for Real-time Processing.
Detailed Issue: The Jetson Nano, while significantly improving the Raspberry Pi in terms of computational power, still has its limitations. When multiple processes run concurrently or when processing high-resolution images in real time, the Jetson Nano might experience performance drops or overheating issues.
Potential Fix: Explore more powerful alternatives like the Jetson Xavier or consider cloud-based processing solutions where data is sent to a server for real-time processing.

7- Dark Parking Lots:
Issue: Inadequate Lighting in Parking Areas.
Detailed Issue: Dark parking lots present a multifaceted challenge. Not only do they make it difficult for camera-based systems to capture clear images, but they also pose safety risks for pedestrians and drivers. Shadows, glare from vehicle headlights, and uneven lighting can further complicate image processing tasks, making license plate detection inconsistent.
Potential Fix: Install additional lighting on the robot to ensure consistent illumination.
Use cameras with enhanced low-light capabilities or infrared functionality to capture more explicit images in dark conditions.

8- ROS Programming Challenges with Topic and DNS Router:
Issue: Communication Breakdowns in ROS.
Detailed Issue: The Robot Operating System (ROS) is designed to facilitate communication between different parts of a robotic system. However, complexities arise

when managing network configurations, especially the Topic and DNS router. If not correctly configured, this can lead to lost data packets, increased latency, or even complete communication blackouts between ROS nodes.
Potential Fix:
Configure the DNS router to ensure proper communication within the ROS framework.
Correctly set up the topic in the 'etc.' file and export the ROS master, ensuring each port knows its corresponding topic.
Regularly review and optimize network configurations related to ROS Topic and DNS settings.

11. References

1- Wiggers, K. (2017, April 27). Knightscope's security robots: What they are, and how they work. Digital Trends. Retrieved March 12, 2023, from https://www.digitaltrends.com/cool-tech/knightscope-robots-interview/

2- S. (2022, September 8). A robot works in the parking lot instead of a security guard. SUNDRIES. https://sundries.com.ua/en/a-robot-works-in-the-parking-lot-instead-of-a-security-guard/

3- Radečić, D. (2022, March 2). How to Detect License Plates with Python and YOLO. Better Data Science. https://betterdatascience.com/detect-license-plates-with-yolo/

4- YOLO Integration with ROS and Running with CUDA GPU. (2021, April 6). Robotics Knowledgebase. https://roboticsknowledgebase.com/wiki/machine-learning/ros-yolo-gpu/

5- Motor and servo configuration. Motor and Servo Configuration - Rover documentation. (n.d.). https://ardupilot.org/rover/docs/rover-motor-and-servo-configuration.html#rover-motor-and-servo-configuration

6- ESP32 WIFI telemetry. ESP32 WiFi telemetry - Copter documentation. (n.d.). https://ardupilot.org/copter/docs/common-esp32-telemetry.html

7- RPLidar A2 360 degree lidar — Copter documentation. (n.d.). https://ardupilot.org/copter/docs/common-rplidar-a2.html

8- Simple Object Avoidance Copter documentation. (n.d.). https://ardupilot.org/copter/docs/common-simple-object-avoidance.html

9- NVIDIA Jetson Download Center. (2023, August 1). NVIDIA Developer. https://developer.nvidia.com/embedded/downloads#?search=Jetson%20Nano%20Developer%20Kit%20Carrier%20Board%20Specification

10- dfrobot.com. (n.d.). 8 Megapixels USB Camera with Microphone (Compatible with Raspberry Pi/ LattePanda/ Jetson Nano). https://www.dfrobot.com/product-2188.html

11- dfrobot.com. (2022, June 5). RPLIDAR A1M8-R6 - 360 Degree LiDAR Laser Range Scanner (12M). https://www.dfrobot.com/product-1125.html

12- dfrobot.com. (n.d.-b). Metal DC Geared Motor w/Encoder - 12V 122RPM 38Kg.cm. https://www.dfrobot.com/product-1210.html

13- PiXHawk 6C - Holybro Docs. (n.d.). https://docs.holybro.com/autopilot/pixhawk-6c

14- L298N Motor Driver Module. (n.d.). Components101. https://components101.com/modules/l293n-motor-driver-module