Obstacle Avoidance and Path Following of An Autonomous Driving Robot

David Ma
Jiawei Lin
Jiayi Wang (Jay)
Zexuan Li
Zhongxian Chen

A project report submitted for the partial fulfillment for the course of ME235 Design of Microprocessor-Based Mechanical Systems

Professor: George Anwar
GSI: Nayesha Gandotra

University of California, Berkeley
May 2023

Contents

1 Background

Cargo transport helps get goods from one place to another inside the warehouse, which is key for running the business involving goods. But, there are some problems in this field. It costs a lot to buy the right equipment and hire skilled people. These costs can be too much for smaller businesses.

Also, setting up and running this equipment needs technical skills. This can be a big hurdle, as not every business has access to people with these skills. Plus, current methods aren't always good at dealing with sudden problems, like unexpected obstacles. This can lead to delays and extra costs.

Lately, there have been some improvements in robotics and automation that could help solve these problems. Robots can be designed to be easy to set up, cost-effective, and able to adapt to changes. But, many of the robot solutions out there are still too expensive or complex, so not everyone can use them.

Another thing that often gets overlooked is making robots easy to use. A robot could be really advanced technology, but if it's not user-friendly, people won't want to use it.

With all these challenges in mind, the team decided to start this project. The idea was to build a robot for cargo transport that is advanced, but also easy to use and affordable. The team used a mix of different technologies and techniques, like Nvidia's Jetbot platform (*JetBot*), OpenCV for image processing (OpenCV 2023), the A* pathfinding algorithm, Model Predictive Control (MPC), a LabVIEW-based GUI, and ultrasonic sensors(Alexis 2023) on an ESP32 chip for real-time interruption.

The goal wasn't just to make a robot that could do the job, but also to make one that's easy for people to use. This background is the starting point for the project. The next sections will go into more detail about how the robot was designed, built, and tested. The project is based on the capstone project "Control a JetBot", corresponding documentation and repository can be found at the end of this report.

2 Motivation

Cargo transport today requires substantial investment in equipment and the hiring of skilled individuals for setup and problem-solving. This can represent a significant cost for businesses. Moreover, these methods often struggle to quickly adapt to unexpected situations, such as sudden obstacles.

The team's goal is to develop a robot that is both easy and quick to deploy, even for users without too much technical knowledge. This approach could save considerable time and money. The team's robot is designed to be robust, and capable of managing unexpected interruptions (i.e. suddenly emerged obstacles). To maintain affordability, expensive components like LiDARs will not be utilized, but the effectiveness of the robot should not be compromised.

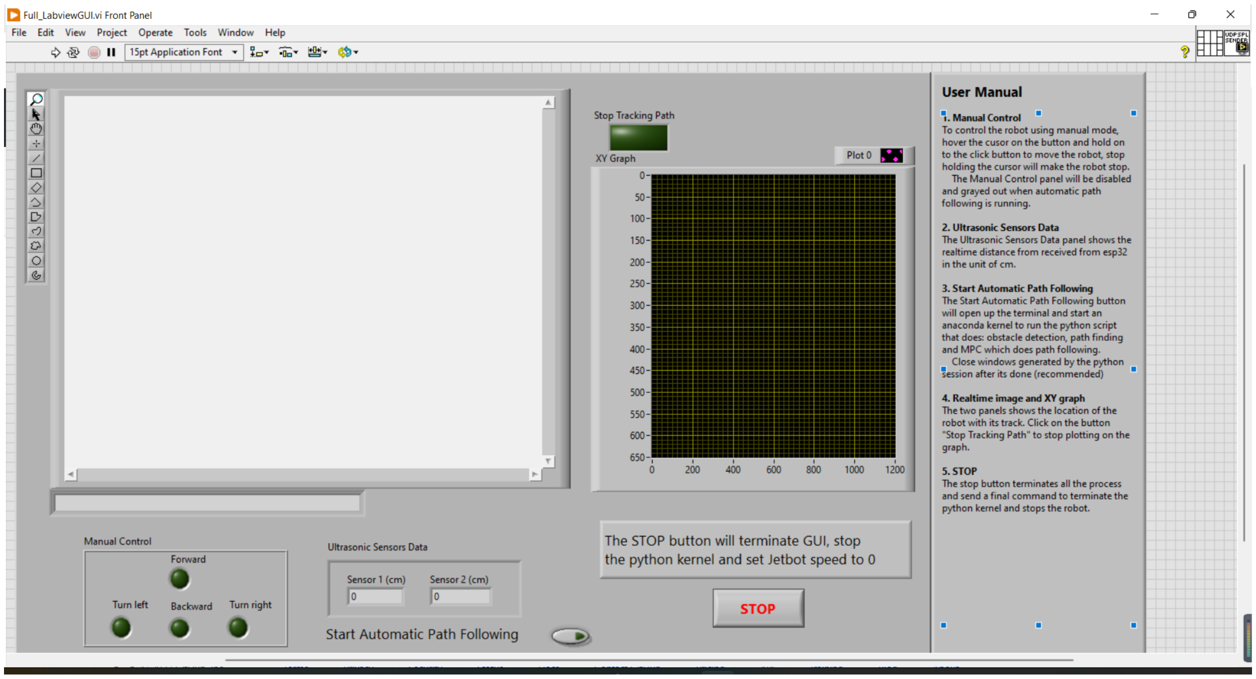
For the ME235 class project, the team intends for the robot to be user-friendly. A straightforward and intuitive graphical user interface (GUI) will be developed for this purpose. Additionally, the robot should be designed to demonstrate multitasking features and respond promptly to interruptions.

In summary, the team aims to enhance the efficiency and affordability of cargo transport with this robot. While maintaining technical proficiency, the robot will also prioritize ease of use for the user.

3 Fulfilling Key Requirements

This section is used to demonstrate how the project is addressing the key requirements of the class. The subsections are divided into 3 parts, GUI, Multi-tasking, and Real-time.

3.1 GUI



The STOP button will terminate GUI, stop the python kernel and set Jetbot speed to 0

User Manual

1. Manual Control
To control the robot using manual mode, hover the cusor on the button and hold on to the click button to move the robot, stop holding the cursor will make the robot stop.
The Manual Control panel will be disabled and grayed out when automatic path following is running.

2. Ultrasonic Sensors Data
The Ultrasonic Sensors Data panel shows the realtime distance from received from esp32 in the unit of cm.

3. Start Automatic Path Following
The Start Automatic Path Following button will open up the terminal and start an anaconda kernel to run the python script that does: obstacle detection, path finding and MPC which does path following.
Close windows generated by the python session after its done (recommended)

4. Realtime image and XY graph
The two panels shows the location of the robot with its track. Click on the button "Stop Tracking Path" to stop plotting on the graph.

5. STOP
The stop button terminates all the process and send a final command to terminate the python kernel and stops the robot.

Figure 1: GUI

As shown in figure 1, Graphical User Interface (GUI) is one of the key requirements for the project. GUI is where the users could have instructional-friendly interaction with software and applications. In the project, users could perform manual and autonomous control by pressing specific buttons on the GUI. The GUI has three main functions on it: user manual control, MPC optimal pathfinding, and path tracking.

The User Manual Control panel is shown in the left bottom corner. It is where the users could press the direction buttons to perform forward, backward, and sideward operations. The operations are accomplished by initializing motor control by setting up GPIO pins and setting motor speeds for different operations in Python.

The optimal pathfinding button is located in the bottom center. The optimal pathfinding for the MPC is achieved by implementing the A* algorithm. This algorithm effectively determines the shortest distance path for the robotic vehicle to reach its destination. By clicking the "Start Automatic Path Following" button, the terminal will open and initiate an Anaconda kernel to execute the Python script responsible for obstacle detection, pathfinding, and path following.

The path-tracking will be shown in the live-stream video player. Pressing the Stop Tracking Path button, indicated by a green LED, will pause the path-tracking. The path-tracking feature in the project allows users to monitor the motion of the Jetbot with ease(Tóth 2012). The GUI utilizes the vision package in LabVIEW to provide a clear visualization of the Jetbot's movements. The GUI displays a live-stream video feed and tracks a set target mark, which is a black circle in this case. Using image processing techniques in LabVIEW, the target mark is detected in each frame of the live-stream video. The GUI then highlights the mark with a red box and plots its location on an XY graph, allowing users to visualize the overall path of the Jetbot as it travels.

The STOP button terminates the LabView process and interrupts the Python kernel, it also mimics the try-catch-finally structure. When the button is pressed, it will first try to terminate the auto path following the process and also sends an instruction to the robot to let it stop.

3.2 Multi-tasking

The multi-tasking part is achieved by the essence of LabView in this project, i.e., there are multiple loops running in parallel in the block diagram. A detailed explanation follows below:

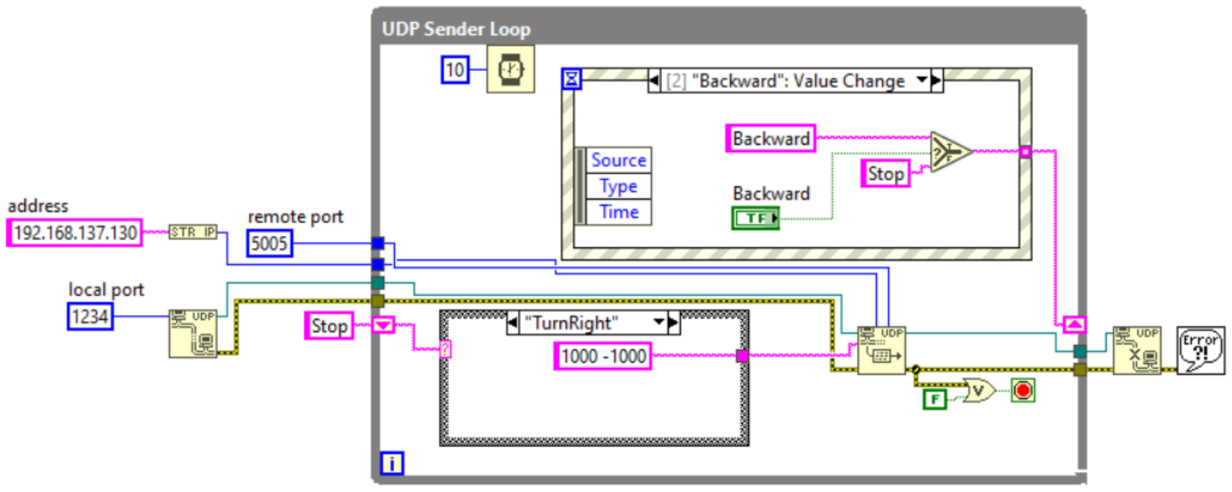
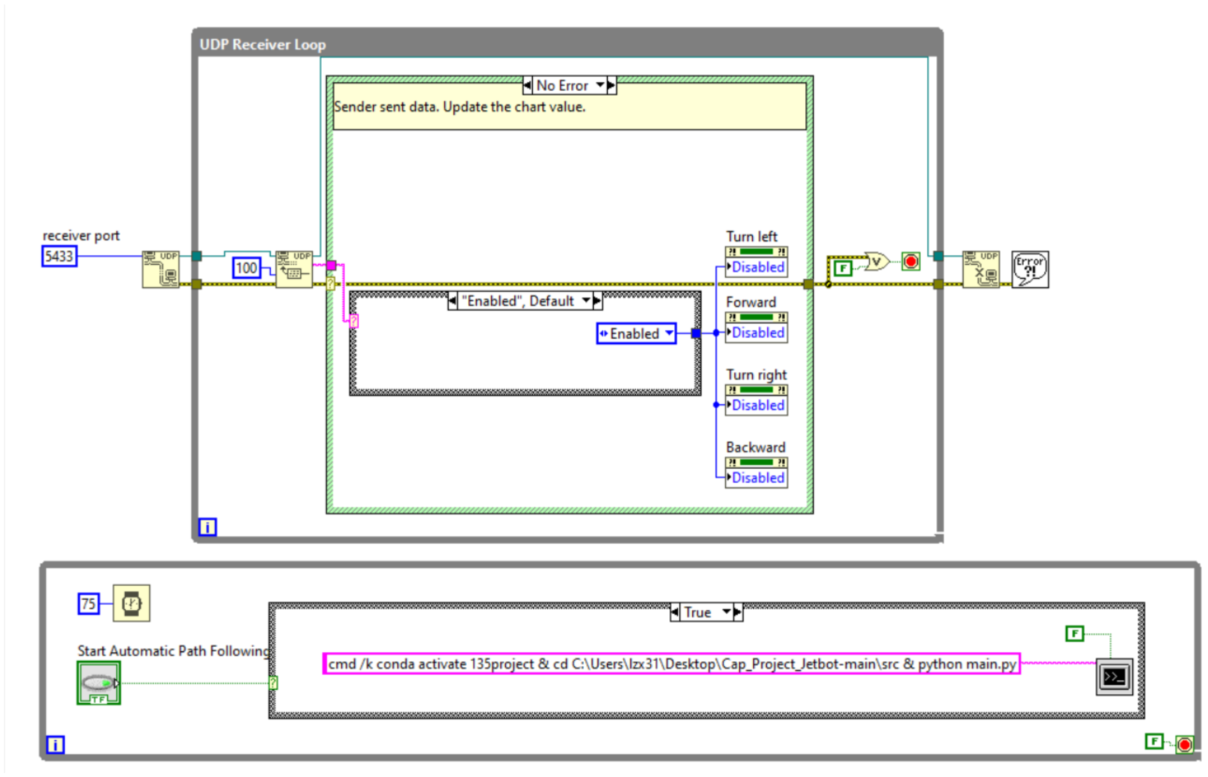


Figure 2: Manual Control Loop

As shown in figure 2, the Manual Control Mode on the front panel is achieved by a UDP sender loop. While no button is pressed, the loop will wait until the state changes. When a button is pushed and held on, the loop sends instructions to the robot continuously until the button is released, LabView sets the wheel speed to 0 and waits for the next command.



```
cmd /k conda activate 135project & cd C:\Users\lzx31\Desktop\Cap_Project_Jetbot-main\src & python main.py
```

Figure 3: Receiver loop for disabled and automatic path following loop

As shown in figure 3, the loop on the top is used to disable and gray out the manual control panel when the automatic path following mode is activated. This is achieved by opening a UDP receiver port listening to the Python kernel, when python code is activated, it sends "disabled" to the LabView and grey things out to prevent any mis-usage.

The loop in the lower half of the image is used to execute the Python code which contains the logic for image processing, pathfinding, and MPC. It will open the terminal and activate the conda environment, then change the directory to the specific location then execute the Python file. Thus, the machine has to have all dependencies installed. Detailed documentation could be found at the end of this report.

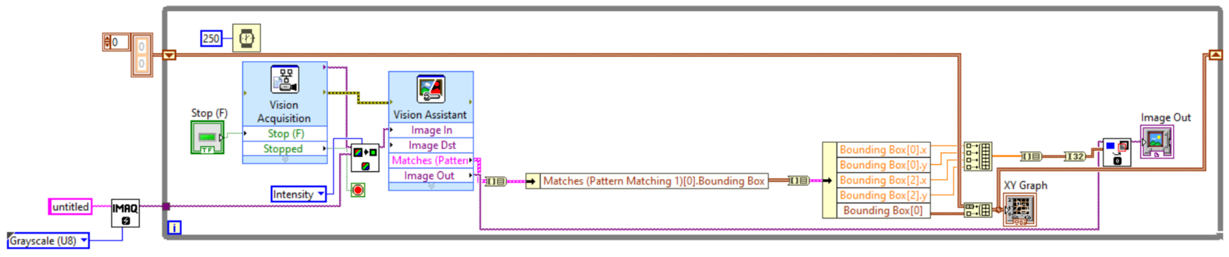


Figure 4: LabView Image Processing Loop

As shown in figure 4, this loop samples the image from the camera and project the real-time video on the canvas on the left-hand side. It also does some image processing: identifies the black dot on the back of the robot and records the path of the robot. The trajectory will also be displayed on the right-hand side.

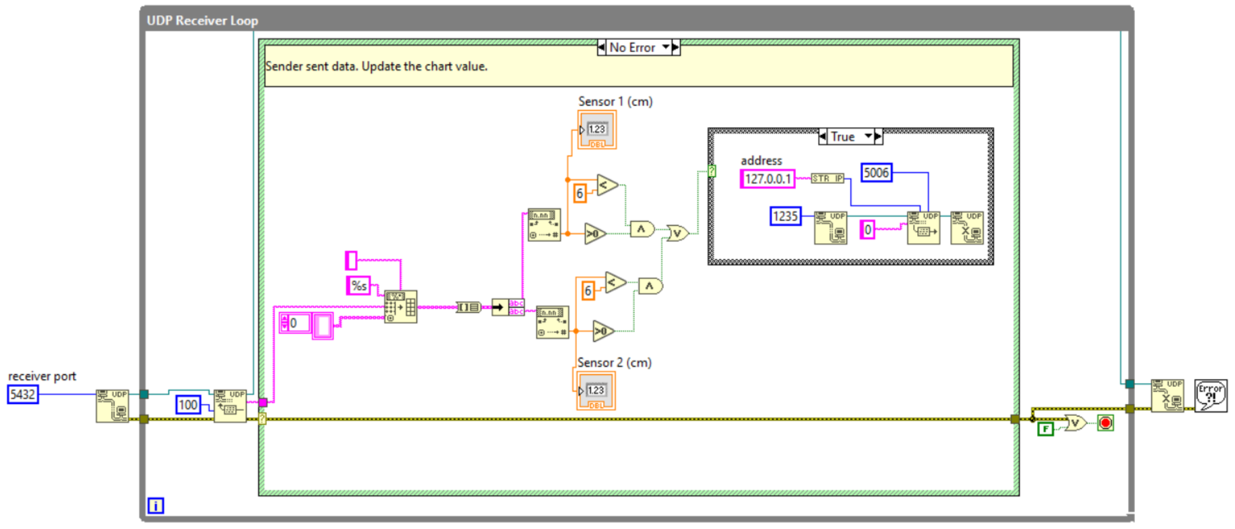


Figure 5: Ultrasonic Sensor Interrupt Loop

As shown in figure 5, this loop is a UDP receiver loop that takes the sample data sent by the ESP32. The distance between the obstacle and the sensors are displayed on two string indicators. It also tackles the logic: when the distance is below a certain threshold, it will send stop instruction to python kernel and terminate the process, also stops the robot.

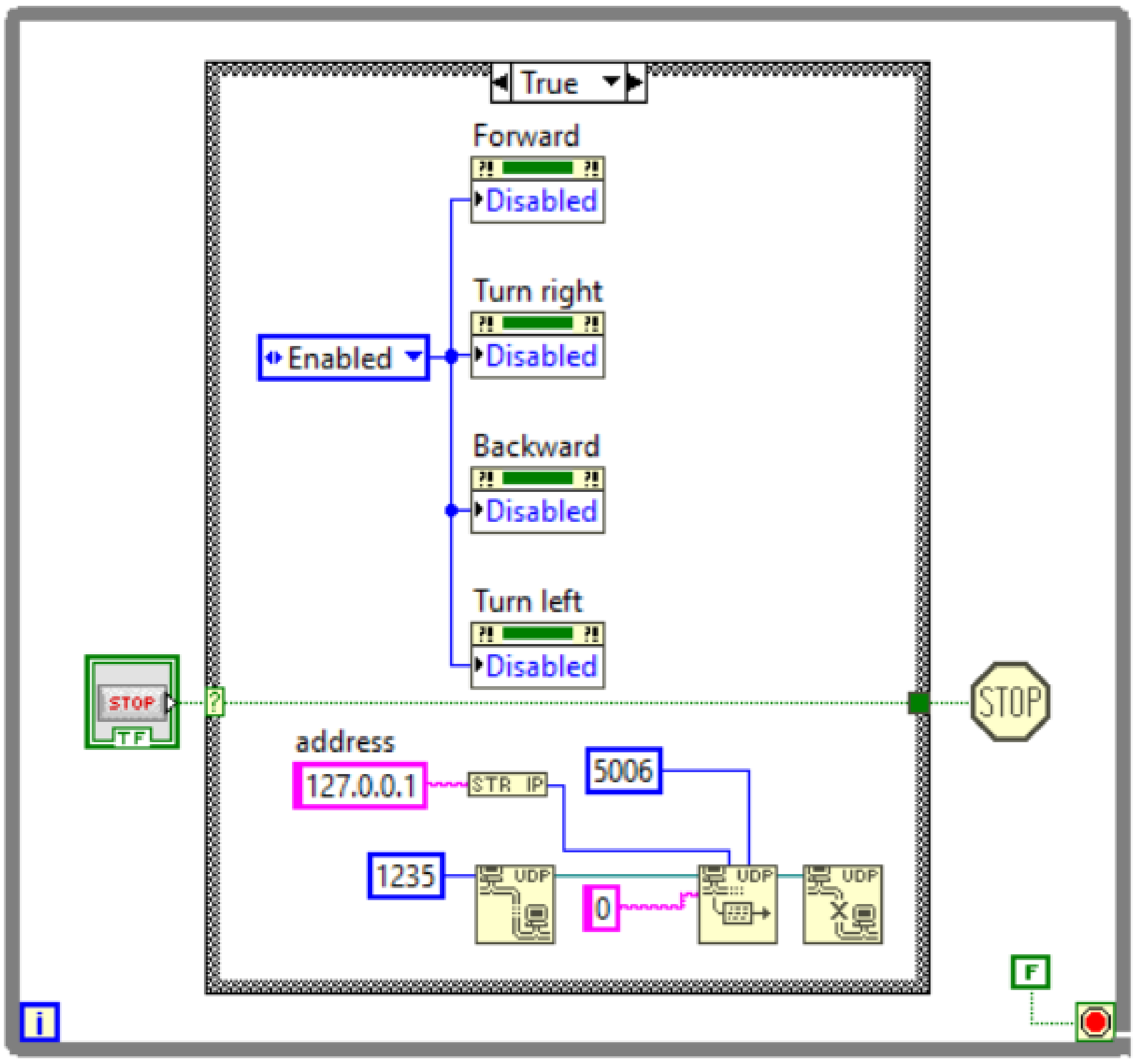


Figure 6: Mimic Try-Catch-finally

As LabView (figure 6) does not support try-catch-finally structure, a customized way of achieve the similar function was figured out. When the STOP button is pressed, the robot will be stopped before terminating the LabView process.

3.3 Real-time

The real-time aspect of the project is mainly demonstrated by obstacle detection. After starting the LabVIEW program, the 2 ultrasonic sensors would be collecting data automatically. The sampling time for object detection is set to be faster than the sampling time for the MPC algorithm, so Jetbot will be stopped once an object is detected. When an ultrasonic sensor detects an object within 15 cm, it should trigger a hardware interrupt. This interrupt immediately interrupts the ongoing task and initiates the necessary actions to stop the car. By employing interrupts, the system minimizes delay and ensures a quick response.

4 Future Work

During the development of Jetbot, several limitations such as time constraints, funding issues, and technical support challenges were encountered, leaving significant room for improvements. In Jetbot 2.0, the primary focus would be to eliminate these limitations and expand its capabilities in various aspects. Firstly, the constraints imposed by the range of activities and hardware would be overcome. Currently, Jetbot is confined to performing MPC optimal pathfinding within a limited range defined by four calibrated markers. To make it suitable for real-world applications, these constraints need to be eliminated. Moreover, the target markers used for Python image processing and LabVIEW are different, necessitating the conversion to a unified marker system.

In terms of hardware enhancements, Jetbot would benefit from the addition of two ultrasonic sensors—one placed at the front and another at the back. This addition would enhance its sensing capabilities and improve obstacle detection. Furthermore, the wheels require replacement, as the current tires tend to dislocate during sideward operations, hindering smooth navigation. Upgrading the wheels would ensure better stability and maneuverability for Jetbot.

Most significantly, the current reliance on a power bank as the energy source for Jetbot's battery needs to be addressed. To achieve sustainable development, it is crucial to modify the power source to utilize solar power. This change would enhance the long-term operational capabilities of Jetbot, allowing it to operate for extended periods without the need for manual recharging.

Overall, Jetbot 2.0 aims to overcome the limitations of its predecessor by eliminating activity range boundaries, unifying target markers, incorporating additional ultrasonic sensors, upgrading the wheels for improved performance, and implementing a solar power system for sustainable energy supply.

5 Reflections and Improvements

Throughout the development of our project, we have gained valuable insights and identified areas for improvement. One key reflection is the significance of cost-effectiveness and ease of deployment in automated factories. By utilizing cameras and DR code vision recognition instead of onboard expensive sensors, we have created a low-cost solution that can be easily implemented. However, further optimization is needed to ensure robust performance in varying lighting conditions.

Another area for improvement is the integration of machine learning for real-time obstacle avoidance. While our current prototype relies on ultrasonic sensors, incorporating an onboard front-view camera and training the system with machine learning algorithms would enable adaptive obstacle detection and avoidance, enhancing autonomy and safety.

Regarding path-following behavior, our implementation of Model Predictive Control (MPC) has shown promising results. However, fine-tuning the controller parameters and optimizing the prediction horizon would enhance the robot's precision and overall performance. One current problem we have identified is the incorrect determination of the front and back of the robot when obstacles are detected. In certain scenarios, the algorithm mistakenly identifies the tail of the robot as the front if it is closer to the planned trajectory. The algorithm needs to be updated to enable the robot to make appropriate decisions based on the correct understanding of its front and back, ensuring efficient and safe navigation.

6 Documentation and Repository

The project is based on the capstone project "Control a JetBot", a corresponding repository can be found here. Documentation could be found here.

The repository for this project could be found **here**.
The video demo could be found here.

References

Alexis. 2023. "Ultrasonic sensor HC-SR04: Tutorial on ESP32 with MicroPython." *uPesy* (January 4, 2023). https://www.upesy.com/blogs/tutorials/hc-sr04-ultrasonic-sensor-on-esp32-with-micropython-tutorial.

JetBot. https://jetbot.org/master/.

OpenCV. 2023. *Home - OpenCV,* May. https://opencv.org/.

Tóth, János. 2012. *LabVIEW USB Cam + Image Processing + MATLAB (MathScript) Tutorial,* March. https://www.youtube.com/watch?v=aUfRAg-rZE4.