NEW YORK UNIVERSITY

ENGR-UH-3530 - Embedded Systems

Final Project Report

Group Members:

Aya, El Mir - ae2195

Kirubel Solomon Tesfaye - kt2350

Lukelo Luoga - ltl2113

December 13, 2023

Introduction

The Embedded Systems Class Project revolves around the development of an autonomous robot inspired by the RoboCupJunior Rescue Line competition rules of 2018. The primary objective is for the robot to autonomously follow a black line on a modular field, overcoming challenges presented by diverse tile patterns, ramps, and environmental conditions. The project emphasizes experiential learning, with the different student teams expected to derive valuable insights from the design, construction, and programming of their robots. In this report, we provide a detailed explanation of our project, our solution, and design choices.

Background Research

The project draws inspiration from the RoboCupJunior Rescue Line competition, where autonomous robots navigate a dynamic field composed of modular tiles with concealed patterns. The unpredictable nature of the tile arrangements adds complexity to the robot's pathfinding task. The white floor, textured surface, and ramps between tiles simulate real-world challenges, fostering a comprehensive understanding of environmental variability. Moreover, the inclusion of speed bumps, debris, and obstacles, coupled with intersections and dead ends, compels teams to develop sophisticated algorithms for efficient navigation. The project also considers environmental factors such as lighting and magnetic conditions, underlining the importance of adaptability in robotic systems.

Project Objectives

The primary objectives of the project are to develop an autonomous robot capable of following a black line on a white floor through a modular field with distinct patterns. The robot should navigate efficiently and overcome challenges presented by intersections marked with green squares, adapting its path accordingly. Additionally, the robot must recognize and respond to stop signs and environmental conditions, ensuring robust performance in varied scenarios. Our project aims to integrate computer vision techniques, such as image processing and contour analysis using OpenCV, to achieve real-time decision-making and demonstrate a comprehensive understanding of autonomous robotic systems in line with the specified rules and requirements.

Approach Overview

The robot utilizes a camera to capture images and employs computer vision techniques implemented using OpenCV. The image processing pipeline involves converting the captured image to grayscale, applying thresholding to identify the black line on a white floor, and detecting green squares as intersection markers. The line-following logic guides the robot's movement based on the position of the detected line. Additionally, the robot recognizes green squares at intersections, determining whether to make a turn or proceed straight. The code incorporates functions for finding intersections, filtering outliers, and processing stop signs. The robot adapts its actions based on environmental conditions, such as detecting the end of the road. The approach demonstrates a combination of image processing, contour analysis, and decision-making to navigate the robot through a predefined course autonomously. Our solution also includes widgets for real-time visualization of processed images and key parameters.

Implementation

Our implementation for the project employs image processing for autonomous navigation. It utilizes the OpenCV and ipywidgets libraries to process real-time camera images. The primary functionalities include detecting and responding to stop signs (red squares) and green squares, implementing line following logic, and making decisions based on the detected shapes and their positions in the captured images. The robot's movements are controlled in response to specific conditions, such as stopping at a red square or making turns at intersections. The code employs color space conversion, thresholding, contour analysis, and intersection point detection to make decisions about the robot's navigation path. The interface is visualized through Jupyter widgets, providing real-time updates on processed images and relevant information.

1. Line Follow

The line_follow function is a crucial component of the project's image processing and control logic. It begins by converting the input image from color (BGR) to grayscale, simplifying subsequent processing. A manual thresholding technique is then applied to create a binary image, differentiating between regions of interest and the background. The function proceeds to identify contours, representing the boundaries of the white regions in the binary image, using the cv2.findContours function. The largest contour, indicative of the detected line, is determined based on its area. The centroid of this contour is computed using moments, and a circle is drawn at this centroid on the original image. Depending on the x-coordinate of the centroid, the robot is commanded to move forward, turn right, or turn left, facilitating line following. Additionally, a forward movement is triggered if the centroid's y-coordinate is below a specified threshold. The grayscale thresholded image and the original image, now annotated with the detected centroid, are then updated in display widgets for real-time monitoring. Overall, this function plays a pivotal role in enabling the robot to autonomously follow a designated line based on visual cues in the captured image.

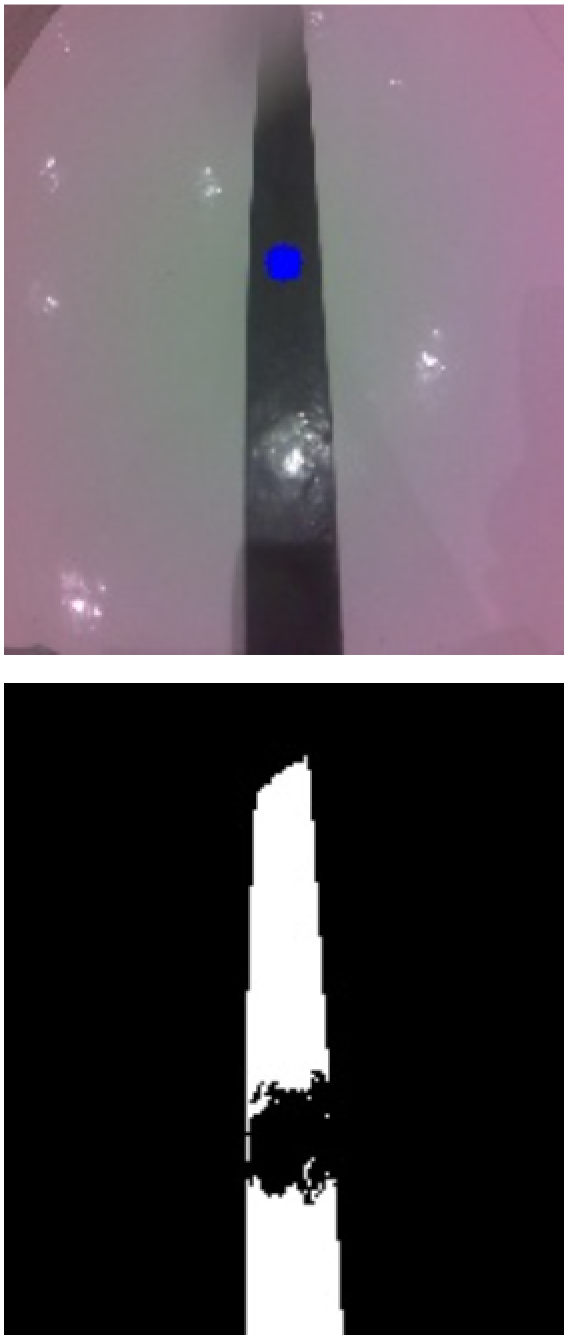


Figure 1: Normal image and black line thresholded image respectively (blue dot = centroid of contour, location to move to)

2. **Green Detection**

The `process_green_square` function is responsible for detecting and analyzing green squares in a cropped region of the captured camera image. It begins by converting the image to the HSV color space and creating a mask to isolate the green color within specified bounds. The contours

of the green squares are then identified, and if the count is less than three, the function defers to the stop sign processing function. Otherwise, it focuses on the largest detected green contour. The function calculates the center of mass (centroid) of the green contour, displaying the x and y coordinates through labeled widgets. It also computes the total area of all green contours. Depending on predefined conditions, including the centroid position and the total area of green contours, the function makes decisions for the robot's navigation. This includes turning at intersections, making U-turns, or continuing forward based on the specific contextual criteria defined in the code. Additionally, visual feedback in the form of a green mask image is displayed in a widget for real-time monitoring.
(add a widget of the green square detection)

3. Intersection Detection

The `find_intersection` function begins by converting the input image to grayscale and applying manual thresholding, followed by Canny edge detection to emphasize prominent features. Utilizing the probabilistic Hough transform, the function identifies line segments in the image. It then iterates through pairs of detected line segments, calculating intersection points using linear algebraic equations. To enhance accuracy, the function filters out outliers based on z-scores, considering points within a specified threshold. The remaining intersection points contribute to the calculation of an average intersection point, serving as an estimate for the center of the intersection. The function returns this average point as a tuple (x, y) or `None` if no intersections are found or if the detected intersections are considered outliers. Overall, this approach aids in the precise identification of intersection points, essential for the robot's navigation decisions in response to the detected lines.

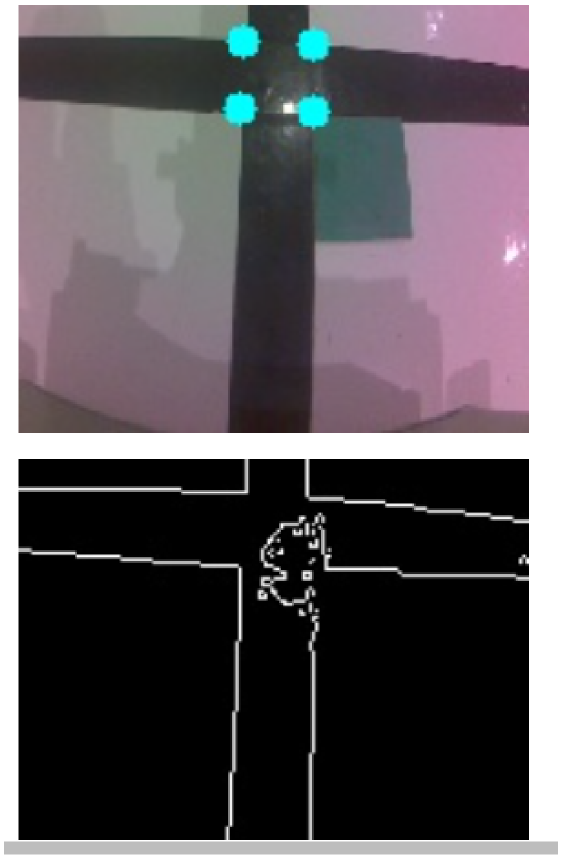


Figure 2: Intersection points (the four blue-green dots are the intersection)

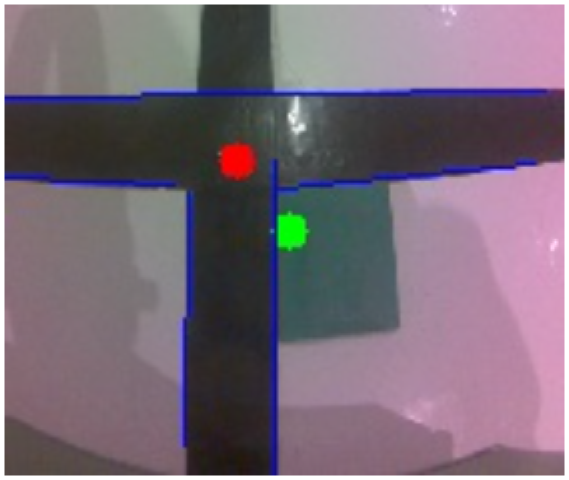


Figure 3: The red point: center of the intersection of the black lines (the four points from Figure 2 averaged). The green dot: location of green. These two dots are compared to find the direction of the turn

4. Stop Sign detection

The process_stop_sign function initially converts the cropped image to the HSV color space. It defines two color ranges corresponding to the red color, creating separate masks for each range. These masks are then combined to form a unified mask representing the red color in the image. The contours of the red squares are identified using OpenCV's contour detection, and if the count of contours is less than three, implying that a stop sign is not clearly detected, the function redirects the processing to the line following logic (`line_follow`).

On the other hand, if multiple contours are detected, the function identifies the largest contour (presumed to be the primary red square) based on contour area. The center of mass (centroid) of this contour is then calculated using the `moments` function. If the centroid is below a certain threshold along the y-axis (`cy > 85`), indicating the bottom part of the image, the robot is instructed to stop, and a message is printed. This decision assumes that the stop sign is located near the bottom of the field of view, simulating a scenario where the robot needs to stop at an intersection. It's important to note that the specific values, such as the color ranges and centroid threshold, may need adjustment based on the characteristics of the environment and the camera setup.

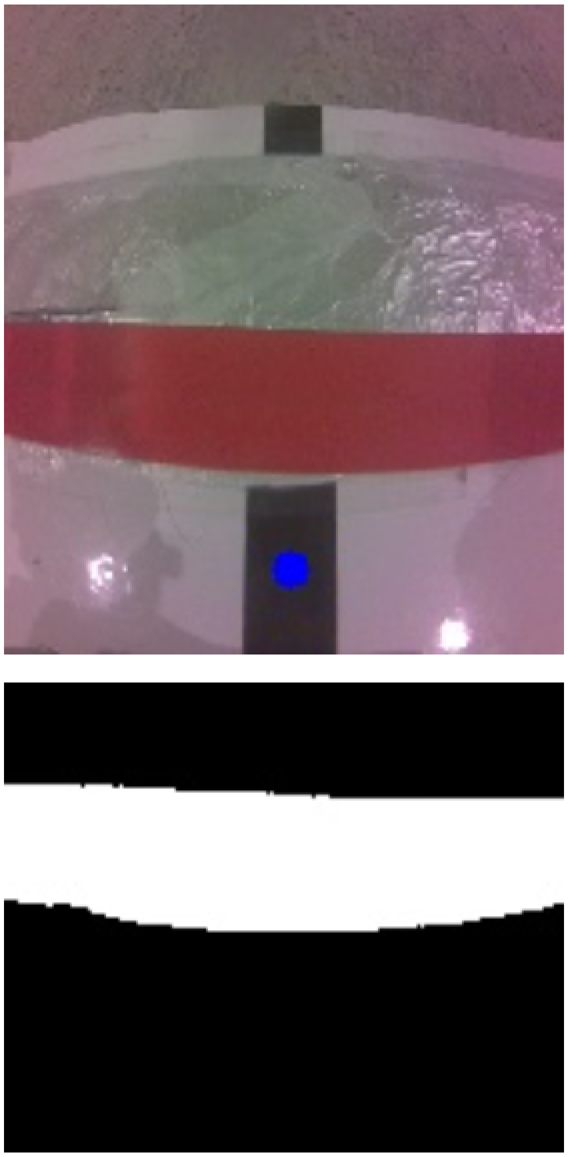


Figure 4: Red line block detection

5. **Update Image**

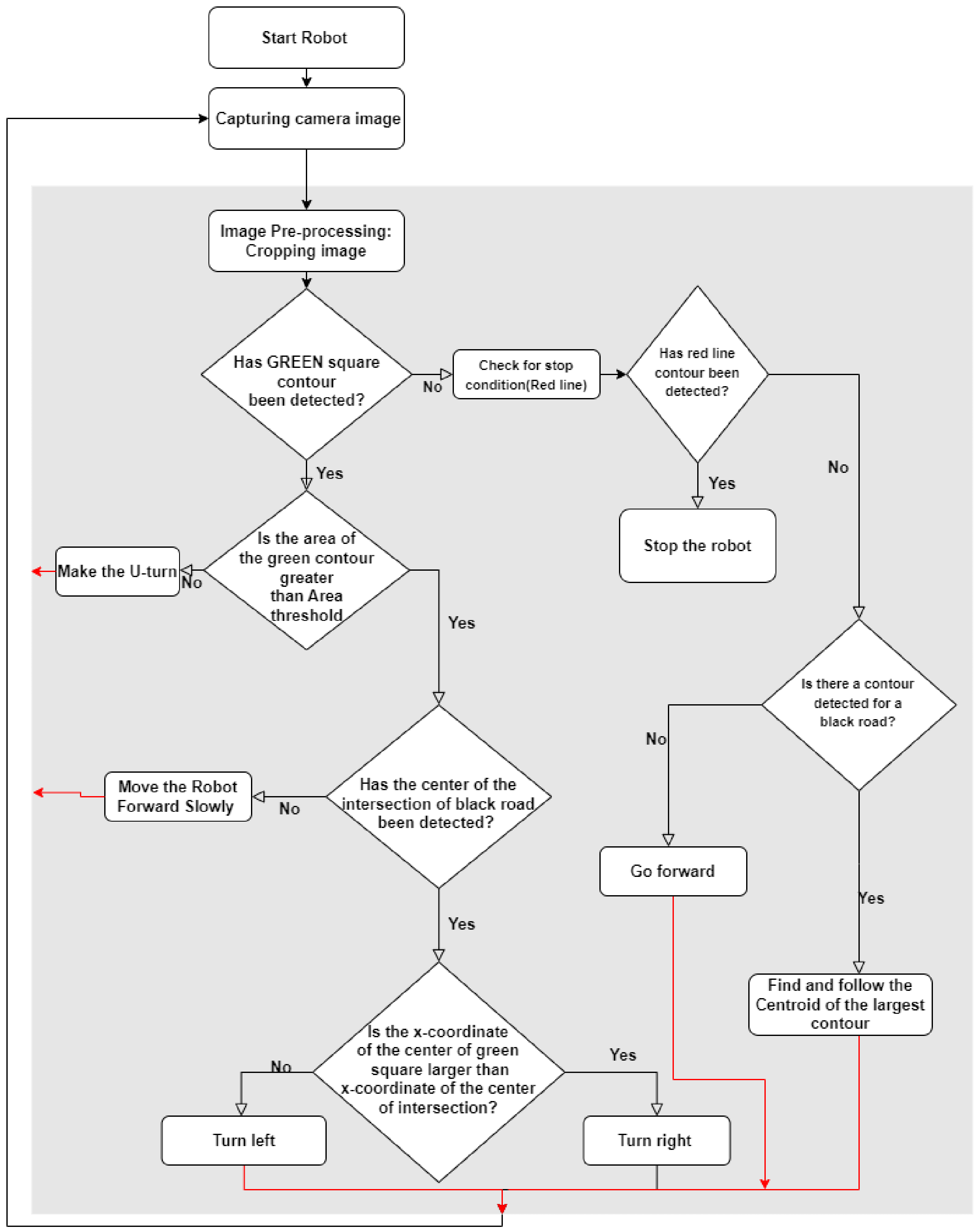
The `update_image` function plays a central role in handling the overall execution of the robot's image processing and navigation logic. This function is linked to the camera and is triggered each time a new image is captured. Within the function, the captured image is cropped to focus on the region of interest. The cropped image is then passed to the `process_green_square` function, which encapsulates the core decision-making logic based on green square detection, intersection analysis, and stop sign recognition. The function internally calls other specialized functions such as `line_follow`, `process_stop_sign`, and `find_intersection` to handle specific aspects of the image analysis.

By modularizing the code in this way, the `update_image` function effectively acts as a higher-level orchestrator, managing the flow of information and decisions during each iteration. The green square detection function, in turn, triggers other functions based on the detected elements in the image. Additionally, the function updates various widgets, such as displaying the green mask, average intersection coordinates, and area labels, providing real-time feedback on the robot's visual perception. Overall, `update_image` handles the coordination and execution of the robot's image processing pipeline, making it a key component in the seamless integration of various functionalities for autonomous navigation.

Note: The robustness of each function has been meticulously tested across diverse lighting conditions to ensure adaptability to real-world scenarios. Since image processing, particularly color-based detection, is sensitive to changes in lighting, adjustments in color thresholds, contour analysis, and other parameters have been made to enhance the code's resilience. Iterative testing under varying illumination conditions has been instrumental in refining the algorithm's performance, making it versatile enough to handle different lighting environments. This comprehensive testing approach ensures that the robot's navigation and decision-making processes remain effective and reliable in a range of real-world settings.

Design Methodology

1. Design Flow Chart



Start Robot
Capturing camera image
Image Pre-processing: Cropping image
Has GREEN square contour been detected?
No
Check for stop condition(Red line)
Has red line contour been detected?
Yes
Stop the robot
No
Yes
Is the area of the green contour greater than Area threshold
No
Make the U-turn
Yes
Is there a contour detected for a black road?
No
Go forward
Yes
Find and follow the Centroid of the largest contour
Has the center of the intersection of black road been detected?
No
Move the Robot Forward Slowly
Yes
Is the x-coordinate of the center of green square larger than x-coordinate of the center of intersection?
No
Turn left
Yes
Turn right

2. Explanation of Flow Chart

Start of the Autonomous Robot Operation:

- The process commences with the initiation of the autonomous robot's navigation system.

Image Acquisition:

- The robot's camera captures a real-time image, serving as the primary input for the robot's vision system.

Image Pre-Processing:

- The captured image undergoes pre-processing where it is cropped to extract the region of interest (ROI) for further analysis. This step is crucial for reducing computational load and focusing the robot's attention on relevant visual cues.

Green Square Contour Detection:

- The system queries whether a green square contour is present in the pre-processed image.
- If no green contour is detected, the robot enters a path-checking phase for a red line.
- If a green contour is detected, the system evaluates its area to determine the robot's course of action:
- If the green contour's area exceeds a certain threshold, this indicates a condition where the robot must execute a U-turn.
- If the contour's area is within acceptable bounds, the robot proceeds to navigate forward slowly, at which point it checks for the center of an intersection on a black road.

Red Line Detection for Stop Condition:

- In a concurrent pathway to green square detection, the system checks for a red line in the image which is interpreted as a stop signal.
- If a red line contour is detected, the robot is commanded to halt its movement immediately, overriding other navigation commands.

Intersection Center Detection:

- If the robot identifies the center of an intersection on the black road, it proceeds to the next decision point:
- The system compares the X-coordinate of the green square's center with the X-coordinate of the intersection's center.
- If the green square's center is positioned to the right of the intersection's center, the robot steers right.
- Conversely, if the green square's center is to the left, the robot adjusts its course to the left.

Forward Movement and Line Following:

- If no intersection is detected, the robot continues to move forward.
- If an intersection is present but no immediate action is required based on the black road's cues, the robot employs a line-following algorithm:

    - It identifies and follows the centroid of the largest contour, presumably a guiding line on the road surface. This behavior ensures the robot maintains its lane or follows a predefined path.

Robot Decision Execution:

- Based on the analysis and decisions made in the previous steps, the robot executes the appropriate navigational maneuvers:
    - Include turning, moving forward, performing a U-turn, or stopping, guided by the visual cues interpreted from the processed image.

Real-Time Image Processing and Reaction:

- The robot continuously captures and processes images, reacting in real-time to visual stimuli based on the logic outlined in the flowchart. This cycle repeats, allowing the robot to adapt its navigation strategy to dynamic environmental conditions.
- The accompanying Python code underpins the described decision-making process. It utilizes the OpenCV library to preprocess captured images, detect contours, identify intersections, and execute the line-following algorithm. Various functions within the code correspond to the different decision pathways in the flowchart, ensuring that the robot's responses to visual cues are both swift and accurate. The robot's movements are thus a direct consequence of the visual processing algorithms, enabling autonomous navigation through its environment.

Lesson Learned

- Intersection Handling: Implemented robust intersection recognition using computer vision for precise navigation based on green markers.

- Stop Sign Compliance: Achieved accurate detection and response to red stop signs through color segmentation in the HSV color space.

- Adaptive Line Following: Developed adaptive line-following capabilities, allowing the robot to efficiently navigate varying line conditions.

- Environmental Robustness: Rigorously tested code under diverse lighting conditions, ensuring the robot's adaptability and consistent performance.

- Real-time Decision-Making: Utilized OpenCV and image processing for real-time analysis, enabling prompt responses to environmental changes.

- Code Optimization: Emphasized code efficiency to minimize response times and enhance overall system performance.

Conclusion

In conclusion, our project successfully addressed the guidelines expected. By leveraging computer vision techniques and OpenCV, we achieved accurate intersection handling, stop sign compliance, and adaptive line following. The integration of real-time decision-making and extensive environmental testing enhanced the robustness of our solution. Looking ahead, further improvements could be made to enhance the system's adaptability in dynamic environments. Implementing machine learning models for more intelligent decision-making, exploring advanced path planning algorithms, and optimizing the code for real-time execution are avenues for future development. Additionally, incorporating more sophisticated sensors and exploring hardware upgrades could contribute to increased precision and efficiency in navigating complex terrains. Continuous testing and refinement will be crucial to ensuring the system's reliability across a broader range of scenarios.

References:

Source Code Github Link: Github Repository

- **https://towardsdatascience.com/getting-started-in-ai-and-computer-vision-with-nvidia-jetson-nano-df2cacbd291c**
- **https://www.inspiritai.com/blogs/ai-blog/2021/8/31/student-blog-project-bridging-ai-and-robotics**
- **https://github.com/thehapyone/Platooning-Robot/blob/master/Robot/Main/Lane%20Detection/lane_detect1.py**
- **https://const-toporov.medium.com/line-following-robot-with-opencv-and-contour-based-approach-417b90f2c298**
- **https://rubikscode.net/2022/06/13/thresholding-edge-contour-and-line-detection-with-opencv/**
- **https://resources.altium.com/p/lane-recognition-and-tracking-nvidia-jetson-nano**
- **https://github.com/chrisdalke/ros-line-follower-robot**
- **https://github.com/NVIDIA-AI-IOT/jetbot**
- **https://towardsdatascience.com/line-follower-robot-using-cnn-4bb4f297c672**
- **https://towardsdatascience.com/getting-started-in-ai-and-computer-vision-with-nvidia-jetson-nano-df2cacbd291c**
- **http://www.roborealm.com/tutorial/line_following/slide010.php**
- **https://ieeexplore.ieee.org/document/8819826**