# ROCKYBORG LINE FOLLOWING AUTONOMOUS ROBOT

Kandanoor Deepika Sahi
56800
Deepika.Kandanoor.odz@fh-zwickau.de

Anna Varghese
57881
Anna.Varghese.oe4@fh-zwickau.de

Muhammad Adil
56266
Muhammad.Adil.off@fh-zwickau.de

Ajith Kathirolil Sabu
56880
Ajith.Kathirolil.Sabu.o9u@fh-zwickau.de

Emilbek Ashyrbekov
57770
emilbek.ashyrbekov.ofs@fh-zwickau.de

Fathima Anakuzhikkara Thekkuveettil
57282
fathima.anakuzhikkara.thekkuveettil.o9k@fh-zwickau.de

Gouri Pavagadhi
55933
Gouri.Pavagadhi.o9x@fh-zwickau.de

Himakumari Patel
56960
Himakumari.Patel.od9@fh-zwickau.de

Jenish Sheladiya
56660
Jenish.Sheladiya.oa2@fh-zwickau.de

Jatin Chawla
55761
jatin.chawla.ogh@fh-zwickau.de

Nandana Sreekumar
56710
Nandana.Sreekumar.oa3@fh-zwickau.de

Ravi Verma
54270
Ravi.Verma.odf@fh-zwickau.de

Shakir Syed Siddiq
57975
Syed.Shakir.ogb@fh-zwickau.de

Sethu Chettukudiyil Sabu
57091
sethu.chettukudiyil.sabu.o9m@fh-zwickau.de

## 1. Abstract

This project focuses on the development of a line-following autonomous robot using RockyBorg hardware and computer vision techniques under the Agile Scrum methodology. The robot was built using a Raspberry Pi 3B+, Pi Camera, and RockyBorg Kit and programmed in Python using OpenCV and the RockyBorg library. Over five months and ten sprints, our multidisciplinary team collaborated to implement line detection, path-following algorithms, and object avoidance mechanisms. Agile practices such as sprint planning, reviews, and retrospectives facilitated clear task distribution and continuous improvement. The final robot successfully detected lines and junctions, handled curves, and responded to red obstacles with consistent results.
**Keywords:** Autonomous Robot, Computer Vision, Agile Scrum, RockyBorg, Line Detection

## 2. Introduction

The Internet of Things (IoT) has paved the way for smarter, more autonomous systems that interact seamlessly with their environments. Among these innovations, autonomous robots have gained significant traction in fields such as logistics, agriculture, and smart mobility. A foundational capability for many of these robots is line-following, which enables them to navigate controlled paths with minimal human intervention. This project aims to build such a robot using the RockyBorg kit, which integrates with a Raspberry Pi 3B+ and supports both motor control, servo motors and vision processing.

Our robot leverages OpenCV for computer vision, enabling the system to detect black lines of curves, junctions, $90^0$ turns. The RockyBorg PCB and camera module were central to integrating hardware and software for autonomous motion. The team adopted the Agile Scrum methodology to manage tasks, prioritize sprints, and ensure continuous improvement throughout the development cycle. This project not only enhances understanding of IoT hardware and software systems but also demonstrates how agile collaboration can drive innovation in robotics.

## 3. Project Objectives

The objective of this project is to develop an autonomous line-following robot using IoT components by applying computer vision for path detection, implementing Agile Scrum methodology in a multi-functional team, and fostering coordination across software, hardware, and quality assurance teams.

## 3.1 Team Roles and Responsibilities

| Role | Members | Responsibilities |
|---|---|---|
| Project Manager | Deepika | Project Planning, Sprint Coordination, Overall Supervision,Report writing |

| | | |
|---|---|---|
| Scrum Master | Anna | Organizing meetings, Sprint meetings,Report writing |
| Development Lead | Ravi | Leading the complete project development of both Software and Hardware |
| Software Team | Adil, Sethu, Ajith, Jenish, and Emilbek | Python, OpenCV |
| Hardware Lead | Siddiq | Leading Hardware Team, Integration and Development |
| Hardware Team | Fatima, Nandana | Rockyborg Assemble, Report writing |
| Testing Lead | Jatin | Leading the Testing team in the Project |
| Testing team | Gouri, Hima | Functional & integration testing,Report writing |

## 4. Methodology

For our IoT project, we followed the Agile Scrum framework to ensure an organized and iterative development process. The entire project spanned five months, divided into 10 focused sprints, each ranging from 4 to 22 days. Our primary milestone: Line Detection, which served as a major checkpoint for tracking progress. A Sprint Gantt Chart was maintained for visualizing timelines and dependencies. All documentation, including sprint plans, task assigning and reports, was managed collaboratively using Google Sheets and Google Drive.

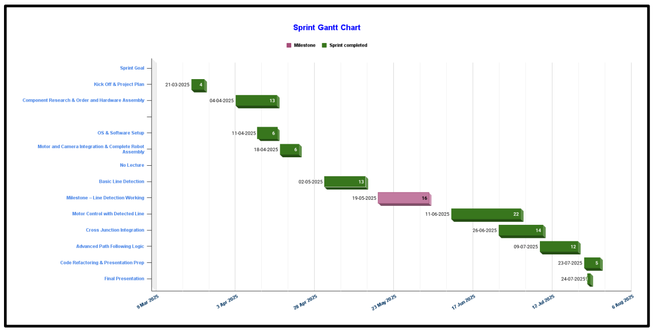


Figure 1: Sprint Gantt Chart

We conducted regular weekly stand-ups to review progress and address blockers, alongside essential Sprint Planning, Sprint Review, and Sprint Retrospective meetings to refine our workflows. The development team tackled a total of 46 tasks, categorized by high, medium, and low priority, while the testing team managed 57 test cases, similarly prioritized to ensure thorough validation. This structured approach ensured continuous improvement and transparency across all phases of our project execution.

## 5. Technical Design

**Hardware Components:**

- Raspberry Pi 3B+
- RockyBorg PiBorg Kit
- Pi Camera Module
- 8 * AAA Batteries
- 32 GB Micro SD Card
- Arduino NANO BLE Sense Rev 2 (Optional)

**Software Components:**

- Python
- C++ (Arduino)
- OpenCV
- RockyBorg Library
- Raspberry Pi OS
- GitHub for version control

## 6. Implementation Details

- Step-by-step development across sprints
- Priority-based task assigning
- Interest-oriented task distribution

The development of the robot was divided into two major milestones: achieving manual control of the robot and basic line detection and then continuing with implementing more advanced path detection of curves, junction, 90° turns and line-following algorithms.

The development of the robot was conducted using a structured approach involving both hardware and software components. The primary goal is to achieve accurate and robust line-following behavior.

### 6.1 Hardware Implementation

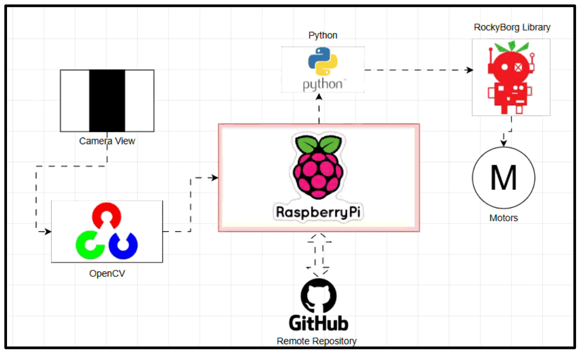


Figure 2: RockyBorg System Architecture

The robot assembly was completed by following the official PiBorg RockyBorg build instructions. The process involved assembling the acrylic chassis, securely mounting the two rear drive motors and the rear servo motor for steering. The Raspberry Pi was installed on the chassis with the RockyBorg PCB mounted on its pins, and the Pi Camera was connected using the CSI cable.

A USB webcam was initially mounted due to delayed procurement of the Pi Camera. The battery pack was installed inside the center console, and all wiring was routed neatly using clips and zip ties. Throughout the build, necessary hardware tools were used for fasteners, and a 3D-printed part was employed to replace a missing component.

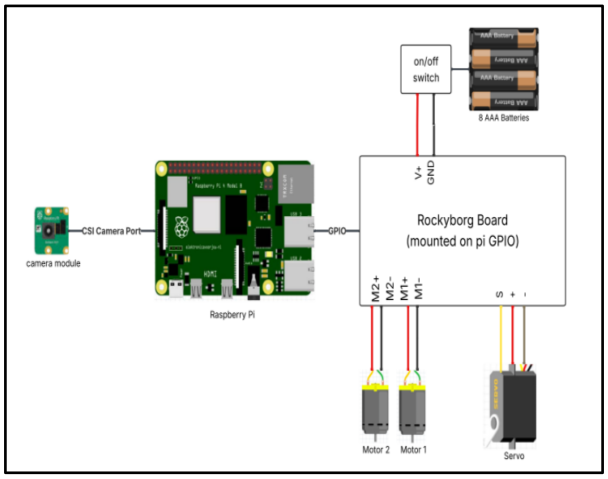


Figure 3: RockyBorg Circuit Diagram

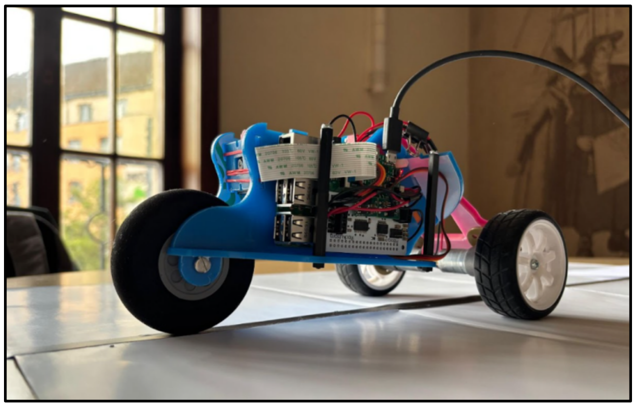


Figure 4: RockyBorg Hardware Build

The servo motor's center position was calibrated using the 'RockyBorg Tuning GUI' program from the RockyBorg Library and was validated using an external leveling device. The RockyBorg library includes Python code ($I^2C$ wrappers) that makes it easier to control both DC motors and servo motors. These wrappers simplify how we send commands to the motors, so we don't have to deal with low-level $I^2C$ communication directly.

The field of view (FoV) and region of interest (RoI) were determined for the camera, optimizing its placement and ensuring a consistent view, crucial for the following algorithms. The hardware setup was tested to confirm operational readiness before integrating the software components.

Regarding the power supply, the robot took the usual 12V powerbank, stepped down to supply only the Raspberry Pi through the micro USB power input port. Another routing was the 8 * AAA batteries, which are capable of supplying current to the complete robot at times.

### 6.2 Software Implementation

The software program was completely developed in Python, utilising the OpenCV library for computer vision tasks such as line detection & image processing. The RockyBorg Python library was used for interfacing with hardware, and the version control was maintained using GitHub for collaborative development.

The initial setup included:

1. Installing Raspberry Pi OS and required libraries
2. Creating a virtual environment
3. Establishing remote access to the Raspberry Pi for concurrent development

Motor calibration was completed by mapping the motor ranges and verifying correct rotational directions. Manual control was achieved using both a keyboard and an Xbox game controller, allowing the robot to be tested before further autonomy.

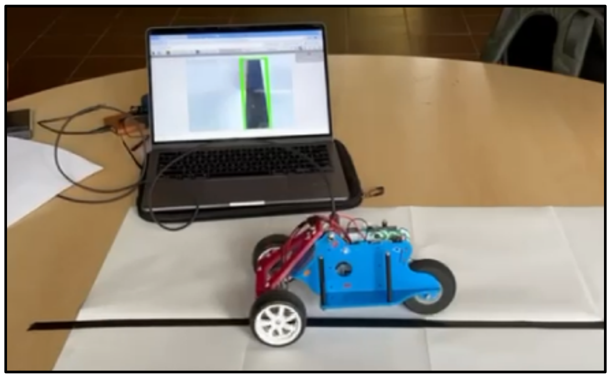


Figure 5: Line-Following Robot in Action

### 6.3 Core Functionalities

The flow of functions of the robot can be understood by the chronological steps described below.

#### 6.3.1 Line Detection

A multi-stage computer vision pipeline was developed using OpenCV to enable accurate line detection:

**Image Capture and Preprocessing:**
Frames were captured using the PiCamera at a resolution of 800×600. Exposure and white balance settings were manually configured to maintain consistent image quality.

**Region of Interest (RoI):**

Image processing was limited to the most relevant portion of the frame where the black line was expected to appear, minimizing surrounding noise and the robot's front wheel.

**Binary Masking (HSV Thresholding):**
The image was converted into the HSV color space. Thresholding was applied to the V (value) channel to isolate the black track, producing a binary image where the track appeared as white against a black background.

**Waypoint Detection and Path Creation:**
The largest white contour in the binary image, corresponding to the track, was identified. Its centroid and slope were calculated, and a series of waypoints were generated. These waypoints were represented as memory overlapping circles that guided the robot's movement, even if the track was briefly not in view.

**PID Steering and Dynamic Speed Control:**
A PID controller was implemented to steer the robot by adjusting the servo angle based on the track's centerline position. Speed was dynamically adjusted using a formula that reduced speed during turns and increased it on straight paths:

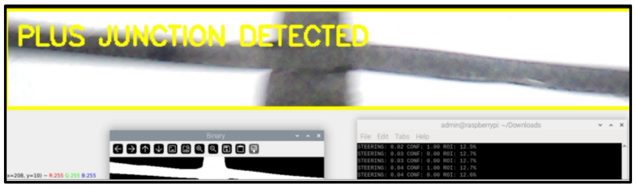
$$m_{speed} = min_s + (max_s - min_s)(1 - |\text{steer}|)$$

### 6.3.2 Curve Handling

Initial tests resulted in overshooting curves due to excessive speed. Speed control was enhanced by applying dynamic adjustment and refining PID constants. These changes improved stability, although minor wobbling persisted on unexpectedly sharp curves.

### 6.3.3 Junction Handling

The robot was trained to identify plus junctions using ROI fill analysis. When a large fill percentage of black pixels were detected in the frame, the robot analyses and performs certain predefined actions which includes locking the steer angle and continuing for *x* amount of frames and straightening up and continuing for *x* more.


PLUS JUNCTION DETECTED
Binary
admin@raspberrypi: ~/Downloads
File Edit Tabs Help
x=208, y=10) ~ R:255 G:255 B:255

```
STEERING: 0.02 CONF: 1.00 ROI: 12.5%
STEERING: 0.03 CONF: 0.00 ROI: 12.7%
STEERING: 0.03 CONF: 0.00 ROI: 12.7%
STEERING: 0.04 CONF: 1.00 ROI: 12.7%
STEERING: 0.04 CONF: 0.00 ROI: 12.6%
```

Figure 6: Junction Detection by fill percentage

### 6.3.4 Red Object Detection

OpenCV's HSV thresholding was used to detect red, green, and blue obstacles. The robot could successfully identify red blocks on the track, halt, and resume after the absence of them.

### 6.3.5 Visualization and Feedback

A visual interface was developed to aid in testing. It displayed the binary mask, RoI and current status messages in terminal such as JUNCTION DETECTED, or SEARCHING FOR LINE, which provided real-time feedback during robot operation.

## 6.4 Techniques Explored During Development

In addition to the core functionality present, several advanced techniques were explored during the development process. While these methods were not fully integrated into the final version, insights gained from testing these approaches contributed to the robustness of the robot's design.

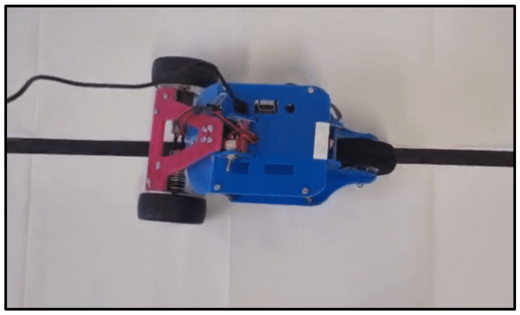


Figure 7: Robot following the black line

### 6.4.1 White Surround Line Following

This approach focused on contour filtering, ensuring that only black contours surrounded by white regions were followed. CLAHE image enhancement and Gaussian-blurred thresholding were used for preprocessing.
Contours were accepted only if the:

- The area exceeded 200 pixels
- At least 30% of surrounding pixels within a 10-pixel margin were white

In ambiguous cases, a fallback search pattern was initiated based on the last known line position.

### 6.4.2 Hybrid VisionRace Strategy

This combined the "multi-slice" image analysis from VisionRace with the robot's own control algorithms. It provided enhanced noise rejection and reliable curve prediction.

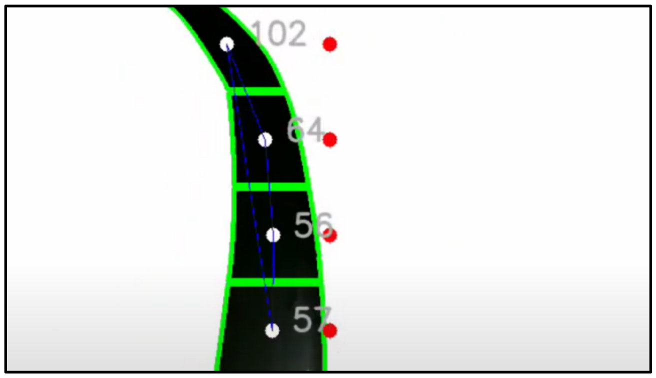


Figure 8: VisionRace Line Following

The primary following black line was divided into <10 sections as shown in figure 8, then two sets of dots are imagined, the red ones represent the center of the camera view and the white ones are the computed centroid of each section. Here the error is logically computed as the lateral difference between the two sets of dots. This method worked well for straight lines and low curves, but it failed in high-radius curves and 90-degree angled turns. A logical follow-up would be to take the Euclidean distance to consider the 90-degree turns, but it turned out to be more complex than expected and was discontinued in the end.

### 6.4.3 Pixel-Binning Technique

A statistical approach to line following, the pixel-binning method, was devised that divided the frame into *x* (default 40) vertical segments. Each bin counted the black pixels in that section. The robot used the weighted mean of active bins to follow the track. This method is computationally efficient *(O(n))*, making it ideal for real-time control on a Raspberry Pi

**Robust Filtering:**

- Weight-based filtering removes outliers
- Retains only bins ≥50% of the pixel count approximately

**Visualization Tools Included:**

- Real-time histograms
- Color-coded bins: Green (active), Blue (low), Red (outlier)
- Threshold indicators and pixel count overlays

Successful test runs showed that the robot could autonomously detect and follow a straight black line and slight curves on a predefined track, with consistent results across multiple trials.

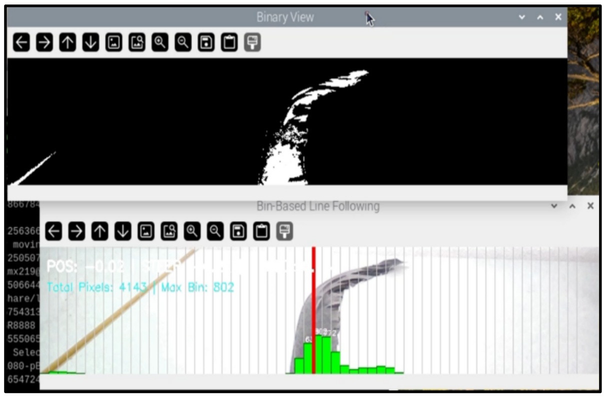


Figure 9: Pixel Binning segmentation

While this algorithm demonstrated excellent performance for straight lines and gradual curves, attempts to enhance it with advanced statistical techniques ultimately hindered its ability to handle extreme maneuvers. The implementation of section-based weighting (outer sections at 30%, inner sections at 100%) and robust outlier filtering as shown in figure 8 using Median Absolute Deviation successfully reduced edge noise sensitivity but created an unexpected problem: during 90-degree turns, when the line appeared predominantly in the outer sections, the algorithm's own protective mechanisms worked against it. The weighted system would discount the very pixels it needed to follow. This created a catch-22 situation where the robot became too focused on the center region, missing critical turning cues that Figure 10: Advanced Pixel Binning visualisation appeared in its peripheral vision. While all of these techniques showed effectiveness in specific cases, they were not adopted in the final implementation, as they were not the most reliable or suitable choice for the overall system requirements.

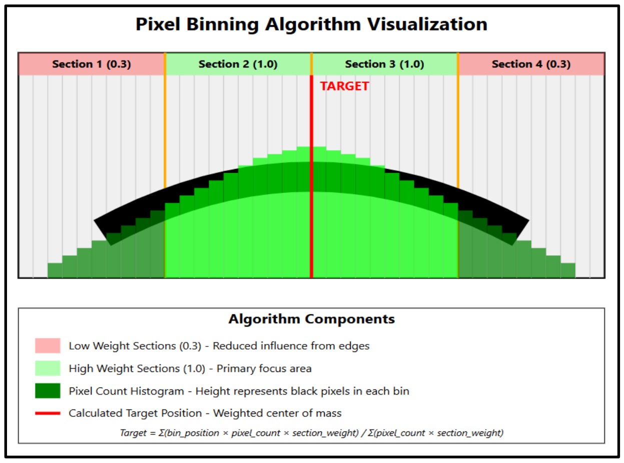

Pixel Binning Algorithm Visualization
Section 1 (0.3)
Section 2 (1.0)
Section 3 (1.0)
Section 4 (0.3)
TARGET
Algorithm Components
🟥 Low Weight Sections (0.3) - Reduced influence from edges
🟩 High Weight Sections (1.0) - Primary focus area
🟩 Pixel Count Histogram - Height represents black pixels in each bin
— Calculated Target Position - Weighted center of mass
Target = Σ(bin_position × pixel_count × section_weight) / Σ(pixel_count × section_weight)

Figure 10: Advanced Pixel Binning Visualisation

### 6.4.3 IMU Visualisation

As an additional feature, the robot was equipped with an Arduino NANO 33 BLE Sense Rev2, which has an in-built Inertial Measurement Unit (IMU) that uses accelerometer and gyroscope readings to understand tilting angles of the robot. Additionally, the board also has a temperature and humidity sensor employed to measure the heat of the Rockyborg PCB and the Raspberry Pi.

As seen in Figure 11, a non-translational digital twin was created to visualise the tilt behaviour of the robot and indicate the current speed and temperature using the appropriate sensors present. After reading the sensor data from the Arduino board through its relevant sensor libraries, the visualisation was made in VPython to make the whole feature quick and not resource heavy.

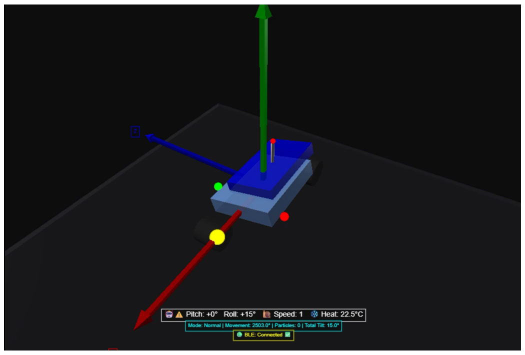


Figure 11: IMU Tilt Visualisation

## 7. Testing

The testing phase was crucial to ensure the RockyBorg robot performed line-following and object detection tasks reliably. Our testing focused on hardware validation, software accuracy and the robot's overall navigation capability on custom-built tracks.

**Hardware Validation:** Power & wiring, checked for correct voltage and secure connections, no issues found.

**Software Testing:** Using OpenCV, the robot was trained to detect a black line via HSV thresholding. We fine-tuned the region of interest (ROI) and used real-time overlays to debug visual processing. A PID controller was integrated to improve steering on curves and reduce oversteer.

**Track Design and Adjustments**: Initial tests using A4 sheets failed due to seams and shadows. Switching to larger 50×70 cm sheets significantly improved consistency. To reduce the glare off the path and improve the robot's vision, we made the track less reflective by using matte paper.

**Corner and Junction Handling:** The robot initially struggled with 90° turns and cross junctions. After implementing predictive angle estimation and memory-based logic, its performance improved at the turns and junctions. We also reduced motor speed and refined PID constants to stabilize the bot during sharp turns and edge line detection.

**Obstacle Detection:** The robot was trained to detect red, green, and blue objects using HSV filters. During tests, it correctly halted for red blocks and resumed after a short delay, meeting the obstacle response requirement.

**Integration Testing:** All core modules-hardware and software were integrated. While most tests passed, minor inconsistencies with camera input under varied lighting remain. Overall, test coverage was high across hardware and software, with ongoing improvements to integration..

**Blockage and Obstacle Handling Issues:**

During testing, one significant challenge was the robot's response to visual blockages and unexpected obstacles on the track.

### 7.1 Issue Observed:

- When a colored object (like a red block) was placed on the track, the robot needed to pause and resume intelligently without misinterpreting it as part of the line or background.
- In some cases, especially on curved sections or during wobbling motion, the robot's vision system would mistake shadows or nearby lines as part of the intended path, leading to incorrect turns or premature halts
- At cross junctions near curves, the bot occasionally confused the junction line with a blockage which triggered the wrong behavior

### 7.2 Technical Challenges:

- There were False Positives due to shadows, lighting changes, or overlapping lines
- Timing delays between detection and response which sometimes caused overcorrection or misalignment
- The camera's limited field of view occasionally failed to provide enough context to distinguish between blockages and path markers

### 7.3 Solutions Implemented:

- HSV thresholding was fine-tuned to detect specific object colors like red
- A (halt and resume) logic was introduced, when a red block is detected the robot stops briefly then continues once the object is out of frame
- Added a smart fallback routine that recalls the last known good path and resumes tracking after a blockage
- Lowered the speed in sensitive areas to improve reaction time

**Camera Setup and Calibration:**

Early in the project, we had problems with the onboard camera as it was not working properly. To avoid further delays we temporarily used a webcam, which let us continue development through Milestone.
We also had a broken camera mount, but instead of waiting for a new part, we designed and 3D-printed a custom fix. This quick solution solved full functionality without slowing us down.
For the Calibration part we went through several rounds of testing to fine-tune the camera's field of view. By adjusting the resolution and region of interest(ROI) we achieved a clear and focused image.

## 8. Results

- Robust Line Detection: Through multi-technique vision pipelines and CLAHE image enhancement
- Smooth Motion: Enabled by PID control, pixel-binning, and weighted ROI
- Accurate Junction Handling: Achieved using memory, contour coverage, and Hough Circle-based waypoint detection
- Efficient Execution: O(n) complexity ensured real-time performance
- Modular Development: GitHub collaboration and component testing simplified debugging and feature extension

## 9. Conclusion

Through this project, we gained hands-on experience in building a real-time autonomous robot with robust line detection, obstacle avoidance, and PID-based motion control. Working in an Agile Scrum environment enhanced our team collaboration and helped maintain steady progress across hardware assembly, software development, and testing. We explored several advanced techniques such as hybrid vision strategy, pixel-binning, and white-surround filtering, which contributed to our understanding of computer vision and real-time embedded systems. The project strengthened our knowledge of Raspberry Pi, OpenCV, and Python programming, while also improving our documentation, version control, and problem-solving skills in a fast-paced, iterative setup. Overall, this project served as a valuable bridge between theoretical IoT concepts and practical implementation.

## 10. Future Enhancements

Future improvements may include Bluetooth/Wi-Fi remote control, cloud-based data logging, integration of ultrasonic or LiDAR sensors for better obstacle detection, and AI-based decision-making algorithms to enhance autonomy in dynamic environments.

## 11. References

- abaeyens. “Image-Processing/RCJ_2014 at Master · Abaeyens/Image-Processing.” GitHub, 2019, github.com/abaeyens/image-processing/tree/master/RCJ_2014. Accessed 2 July 2025.
- Astrom, Karl Johan. PID Control. 2002.
- Carey, Ian. “Arduino PID Controller - from Scratch!” Www.youtube.com, 26 Feb. 2023, www.youtube.com/watch?v=RZW1PsfgVEI. Accessed 2 May 2025.
- CRM-UAM. “GitHub - CRM-UAM/VisionRace at Drive_test.” GitHub, 2017, github.com/CRM-UAM/VisionRace/tree/drive_test. Accessed 20 June 2025.
- “GitHub - CRM-UAM/VisionRace: Python Image Processing Script Based on OpenCV to Enable the Control of a Line Follower Robot through a Camera.” GitHub, 2017, github.com/CRM-UAM/VisionRace/tree/master. Accessed 20 June 2025.
- fustyles.“Arduino/ESP32-CAM_Car/ESP32-CAM_CAR_TrackColorline_Tracking.js/ESP32-CAM_CAR_TrackColorline_Tracking.js.ino at Master · Fustyles/Arduino.” GitHub, 2017, github.com/fustyles/Arduino/blob/master/ESP32-CAM_Car/ESP32-CAM_CAR_TrackColorline_Tracking.js/ESP32-CAM_CAR_TrackColorline_Tracking.js.ino. Accessed 2 July 2025.
- GG popa. “Line Following Robot OpenCV.” YouTube, 6 May 2017, www.youtube.com/watch?v=ZC4VUt1I5FI. Accessed 19 June 2025.
- Hoska, Richard. “Blocked.” Github.com, 2025, github.com/xioTechnologies/Gait-Tracking-With-x-IMU/tree/master. Accessed 29 June 2025.
- Numpy. “NumPy.” Numpy.org, 2024, numpy.org/. Accessed 27 July 2025.
- OpenCV. “OpenCV Library.” Opencv.org, 2019, opencv.org/. Accessed 27 July 2025.
- Out of the BOTS. “Follower Computer Vision Angle Detection Lesson 3.” YouTube, 16 Mar. 2017, www.youtube.com/watch?v=o2ul4KrLT-s. Accessed 2 July 2025.
- piborg. “GitHub - Piborg/RockyBorg: Repository for the RockyBorg Robot Kit Code.” GitHub, 2019, github.com/piborg/RockyBorg/tree/master. Accessed 27 July 2025.
- PYTHON. “Python.” Python.org, Python.org, 2025, www.python.org/. Accessed 27 July 2025.
- “RockyBorg Blue and Pink.” PiBorg, 2019, www.piborg.org/robots-1/rockyborg. Accessed 27 July 2025.
- Technologies, x-io. “XioTechnologies/Gait-Tracking.” GitHub, 27 Apr. 2024, github.com/xioTechnologies/Gait-Tracking. Accessed 30 June 2025.
- Kahan. “GitHub.” YouTube, 17 Mar. 2021, www.youtube.com/playlist?list=PLy4OcwImJzBKzWWb9K_WB3QzaxoiGmxyo. Accessed 27 July 2025.
- Lam, Sahn. “How Git Works: Explained in 4 Minutes.” Www.youtube.com, 28 Nov. 2023, www.youtube.com/watch?v=e9lnsKot_SQ. Accessed 27 July 2025.