Traffic Light Detection in Duckietown

Xue Bingjie, Thorbjörn Höllwarth, Angel Manzano, Himanshu Joshi

20 December 2024

Contents

1 Introduction

This project focuses on developing a modular and autonomous robotic system for Duckiebots, which implemented traffic light detection and response within Duckietown. We implemented **traffic light detection**, **red line detection**, and a **state machine**. To enable the Duckiebot to navigate effectively and make dynamic decisions based on real-time perception, the architecture consists of distinct nodes that communicate via ROS topics and provide essential data for navigation and control.

1.0.1 Nodes and Components Overview

1. **Traffic Light Detection Node**:
   - Utilizes a **YOLO Nano** deep learning model to detect traffic lights within the Duckiebot's camera feed.
   - The detected regions are processed using **HSV-based color filtering** to determine the state of the traffic light (red or green).
   - Results are published to the `/traffic_light_color` and `/traffic_light_size` topics to inform the state machine of the current traffic light signal.
2. **State Machine Node**:
   - Manages the Duckiebot's behavior by transitioning between different states:
     - [**RUN**]: Lane-following mode.
     - [**STOP**]: Halts the Duckiebot when a red line or red traffic light is detected.
     - [**TURN**]: Performs turns at intersections based on inputs.
   - The state transitions are based on data received from the traffic light detection node.
   - Communicates using ROS topics such as `/joystick_override`, `/traffic_light_color`, `/traffic_light_size` and `/wheels_cmd`.
3. **Red Line Detection Node**:
   - Implements red line detection using **OpenCV** for image processing and **HSV color masking** to identify red lines in the Duckiebot's camera feed.
   - Publishes data to `/red_line_detected` topic.

1.0.2 System Integration

The system operates by fusing inputs from the **Traffic Light Detection Node** to dynamically control the Duckiebot's behavior via the **State Machine Node**. The ROS architecture facilitates seamless communication between these nodes, ensuring that the Duckiebot can make real-time decisions. This integration allows the Duckiebot to:

- Identify and respond to traffic light signals.
- Adjust movements based on real-time data from perception nodes.

By combining deep learning techniques and classical computer vision for traffic light detection, the system ensures reliable and efficient navigation for the Duckiebot. Due to time resource limits, we didn't integrate red line detection node with other parts finally. Instead, we used the size of the traffic light's bounding box to control the appropriate stopping position of the Duckiebot at the intersection.

2 Methodology

2.1 State Machine

The state machine serves as the brain of the algorithm, orchestrating transitions between states based on real-time environment feedback. It coordinates critical decisions, such as stopping, turning, and progressing through the track, while implementing traffic light logic to ensure compliance with external signals. By tracking intersections and detecting red lines, it maintains an accurate understanding of the vehicle's position and context. Ultimately, the state machine ensures a smooth and reliable progression through the algorithm, acting as the central framework for decision-making and control.

2.1.1 State Machine design

Our final state machine design is as shown below:

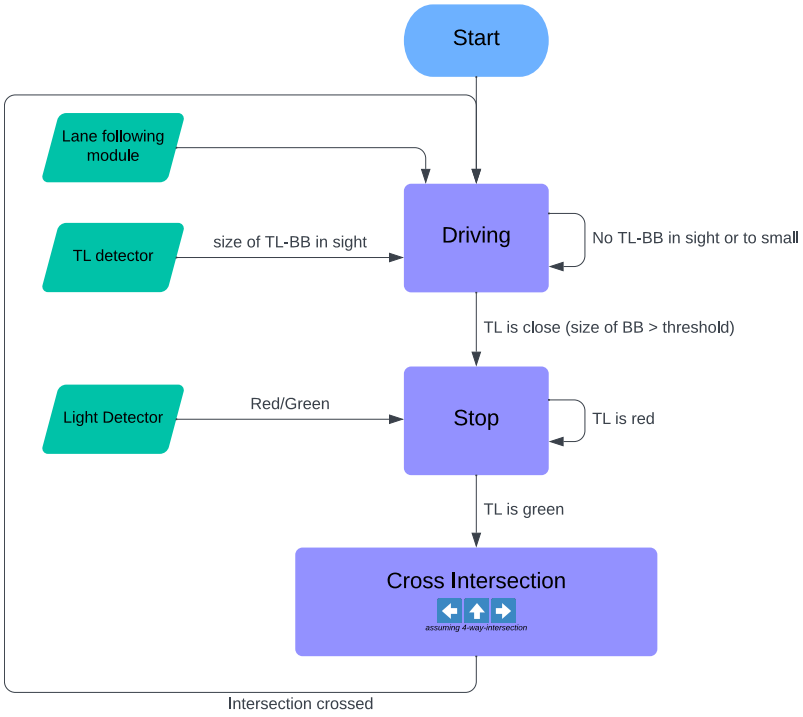


Figure 1: Implemented state machine

2.1.2 State Machine implementation

We implemented the aforementioned state machine using ROS SMACH. Within the state machine, we defined three states: [**RUN**], [**STOP**], and [**TURN**].

Upon activation, the state machine transitions to the initial state, [**RUN**], where the Duckiebot executes the lane following baseline provided by Duckietown. We used the size of the bounding box as a measure to determine whether to stop. When a traffic light bounding box covers a big enough area of the image, which is regarded to be close enough, the Duckiebot transitions to the [**STOP**] state. If a red light is detected, the Duckiebot remains stationary. If a green light is detected, the Duckiebot transitions to the [**TURN**] state and proceeds through the intersection. While traversing the intersection, the Duckiebot makes a random choice among left turn, right turn, or moving straight, and executes the corresponding wheel control commands.

2.2 Traffic light detection

The traffic light color detection algorithm implemented in this project is designed to enable the Duckiebot to identify and respond to traffic lights in the Duckietown environment. The methodology involves a two-step process: detecting the traffic light structure using a YOLO Nano network and determining the light color through image processing techniques.

2.2.1 Structure detection

To identify the location of traffic lights, we use an implementation of the YOLO Nano network [1], [2]. We train the model with a custom dataset consisting of $\sim$ 2000 images of the Duckietown environment. Specifically, the images are taken from all four sides of an intersection with the traffic light always visible, and with the LED either off, red, or green. Each frame is annotated in COCO format with a bounding box around the wooden box that holds the LED light (Figure 2).

Figure 2: Example of an annotated frame use to train the YOLO Nano model for traffic light structure identification.

The model is first trained on a dataset containing images of the Duckietown traffic light with the LED light off. The trained model is then fine-tuned using annotated frames with the LED light on (either red or green). This allows the model to first focus only on the shape and location of the structure. which is agnostic to the color of the light and the direction of travel. Once the model is able to accurately identify the structure, the fine-tuning makes sure the model can focus only on learning about the presence of different colors within the structure.

The final model receives an image as input and outputs a list of bounding boxes. The bounding box with the highest confidence score above a predetermined threshold is chosen, which bounds the detected traffic light structure, as show in Figure 3.

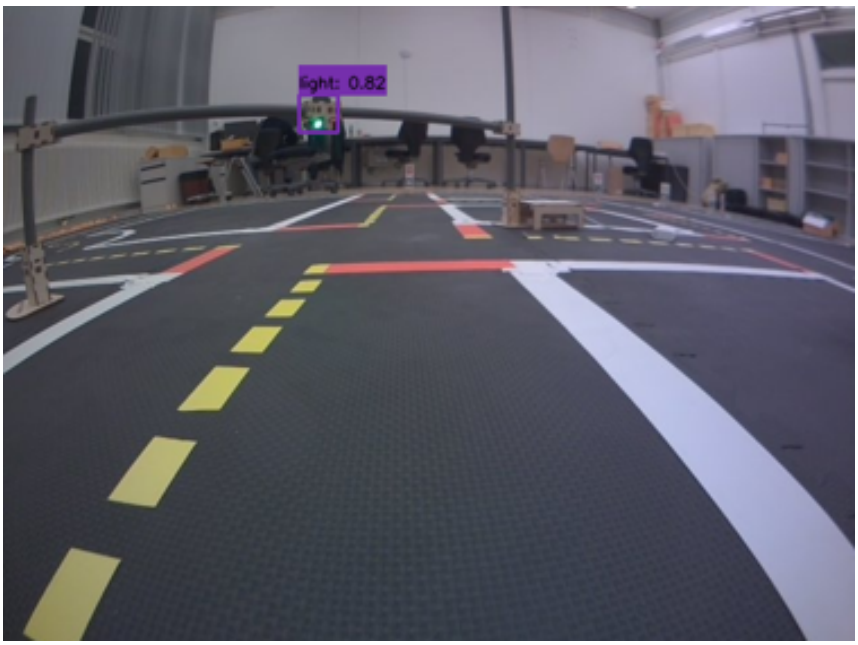


Figure 3: Image input to the YOLO model with the detected bounding box overlaid.

	Lower Bound	Upper Bound
red_1	[175, 18, 204]	[179, 107, 255]
red_2	[0, 18, 204]	[8, 107, 255]
green_1	[66, 50, 188]	[80, 116, 255]
green_2	[80, 50, 200]	[100, 255, 255]
green_3	[144, 37, 235]	[165, 121, 251]

Table 1: Lower and upper bounds for color masking.

2.2.2 Light color identification

With the bounding box of the traffic light, the image is cropped to focus exclusively on this area. This step reduces the influence of irrelevant regions in the image and isolates the traffic light for further processing.

The cropped image is first converted into the HSV color-space for easier processing. It then undergoes color filtering using two distinct masks:

- Red mask: filters the images to highlight light red, dark red, and pink regions while suppressing all other colors.
- Green mask: filters the images to highlight light and dark green regions while suppressing all other colors.

These masks are created by applying color threshold techniques, which involve selecting a range of pixel T-values corresponding to red or green hues in the HSV color space. After applying the masks, the number of non-zero pixels (pixels that pass the filter) is computed for both the red and green masks.

The final step determines the traffic light color based on the mask with the highest count of non-zero pixels. If the red mask contains more nonzero pixels, then the traffic light is considered in red state. Likewise, if the green mask contains more nonzero pixels, then the light is assumed to be green.

This approach ensures a robust and efficient determination of the traffic light color, even in environments with varying lighting conditions and potential background noise.

By combining the strengths of deep learning for structure detection and traditional image processing for color determination, this methodology provides an effective solution for traffic light recognition in Duckietown. The results of this method contribute to the Duckiebot's ability to navigate intersections safely and in accordance with the traffic light signals.

2.3 Red Line Detection

The Red Line Detection module enables the Duckiebot to detect red lines on the track and estimate their distance. This information can provide the state machine information to make accurate decisions, such as stopping at intersections.

2.3.1 Implementation

The module is implemented as a ROS node named `red_line_detector_node.py`. It subscribes to the Duckiebot's camera feed, processes the images, and publishes detection results. The core steps of red line detection are:

- **Image Acquisition**: The node subscribes to the compressed camera feed topic
- **Color Segmentation**: The image is converted to HSV color space, and red regions are detected using two HSV ranges to account for hue wrapping.
- **Noise Reduction**: Morphological operations are applied to reduce noise in the binary mask.
- **Line Detection**: Canny edge detection and Hough Line Transform are used to identify red line segments in the processed image.
- **Distance Estimation**: The distance to the red line is estimated based on the average y-coordinate of the detected lines.

3 Evaluation/Results

3.1 Evaluation methods

3.1.1 Traffic light detection evaluation

To evaluate the performance of the traffic light detection pipeline, a combination of quantitative metrics and qualitative analysis are employed. The evaluation focuses on two key aspects: the accuracy of the traffic light structure detection and the correctness of the color classification.

1. Structure Detection Accuracy: The detection accuracy of the YOLO model is assessed using standard object detection metrics such as Average Recall (AR) and Average Precision (AP):

- Precision refers to the fraction of correctly predicted positive samples (i.e., detected traffic lights) out of all the predicted positives (i.e., all the samples the model detected as traffic lights, whether they are correct or not). **Average Precision (AP)** is the average of precision scores at various levels of recall, typically measured at different Intersection over Union (IoU) thresholds, ranging from 0.5 to 0.95. The IoU threshold defines how much overlap is required between the predicted bounding box and the ground truth bounding box to count as a correct detection. AP is computed by averaging the precision over several recall values (Equation 1).
- Recall refers to the fraction of actual positive samples (i.e. actual traffic lights) that the model correctly detects. **Average Recall (AR)** is the average recall at various IoU thresholds, or it can be computed at different object sizes (e.g., small, medium, large) (Equation 2).

$$\text{AP} = \frac{1}{N} \sum_i \text{Precision at Recall}_i \tag{1}$$

$$\text{AR} = \frac{1}{N} \sum_i \text{Recall at IoU}_i \tag{2}$$

where

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}.$$

These metrics are calculated on a held-out test dataset containing annotated traffic light structures that were not used during training.

2. Color Classification Accuracy: The correctness of the color classification step is evaluated by comparing the predicted traffic light color (red or green) with the ground truth labels. The metric used for this evaluation is the classification accuracy, calculated as:

$$\text{Acc} = \frac{\text{True Red} + \text{True Green}}{\text{Total instances}}. \tag{3}$$

A confusion matrix is also generated to provide detailed insights into the types of errors (e.g., misclassification of red as green and vice versa).

3. Qualitative Analysis: In addition to quantitative metrics, qualitative analysis are performed by visualizing the output of the pipeline on a diverse set of images. This will help identify common failure cases, such as incorrect bounding box predictions or misclassification in challenging conditions (e.g., varying lighting or occlusions).

To evaluate the robustness of the pipeline, the system is tested in real-world scenarios using live camera feeds from the Duckiebot. Performance under different conditions (distance to traffic light, orientation of the Duckiebot with respect to the traffic light, and lit or darker environment light) are recorded and analyzed.

By combining these evaluation methods, a comprehensive assessment of the pipeline's performance and reliability will be achieved, ensuring its suitability for deployment in the Duckietown environment.

3.2 Results

3.2.1 Traffic light detection results

1. Structure detection accuracy **The Average Precision (AP)** is computed at IoU thresholds ranging from 0.50 to 0.95. The results are summarized as follows:

- **AP@[IoU=0.50:0.95] (All sizes)**: The model achieved an AP score of **0.480** across all object sizes. This indicates that, on average, the model correctly detected traffic lights with reasonable precision when considering multiple IoU thresholds.
- **AP@[IoU=0.50] (All sizes)**: At the 0.5 IoU threshold, the AP is **1.0**, demonstrating that the model is completely accurate at detecting traffic lights with a 50% overlap between the predicted and ground truth bounding boxes.
- **AP@[IoU=0.75] (All sizes)**: At the 0.75 IoU threshold, the AP is **0.380**, demonstrating that the model struggles in detecting traffic lights with a 75% overlap between the predicted and ground truth bounding boxes.
- **AP@[IoU=0.50:0.95] (Small objects)**: The performance on small objects is defined by an AP of **0.455**, indicating that the model struggles when detecting smaller traffic lights.
- **AP@[IoU=0.50:0.95] (Medium objects)**: The model showed better performance for medium-sized objects, achieving an AP of **0.655**, suggesting it performs better on objects that occupy a moderate portion of the image.
- **AP@[IoU=0.50:0.95] (Large objects)**: For large objects, the model exhibited the highest precision, with an AP of **1.000**, indicating perfect precision when detecting large traffic lights.

Average Recall (AR) was also computed at different IoU thresholds to assess the model's ability to detect traffic lights across varying precision levels:

- **AR@[IoU=0.50:0.95] (All sizes)**: Between 0.5 and 0.95 IoU thresholds, the model's recall was **0.581**, showing that the model successfully identified a substantial portion of the traffic lights.
- **AR@[IoU=0.50:0.95] (Small objects)**: Recall for small objects was **0.559**, slightly lower than the overall recall at this IoU threshold. This suggests that the model somewhat struggles more in detecting small traffic lights.
- **AR@[IoU=0.50:0.95] (Medium objects)**: Recall for medium-sized objects was **0.680**, which is better than the AR for small objects.
- **AR@[IoU=0.50:0.95] (Large objects)**: The recall for large objects was the highest, at **1.0**, which means that the model successfully detected all traffic lights occupying a large portion of the image across the full range of IoU thresholds.

These results indicate that the model performs better on larger traffic lights, and smaller objects present a more significant challenge. This pattern is common in object detection tasks, where smaller objects tend to be harder to localize accurately.

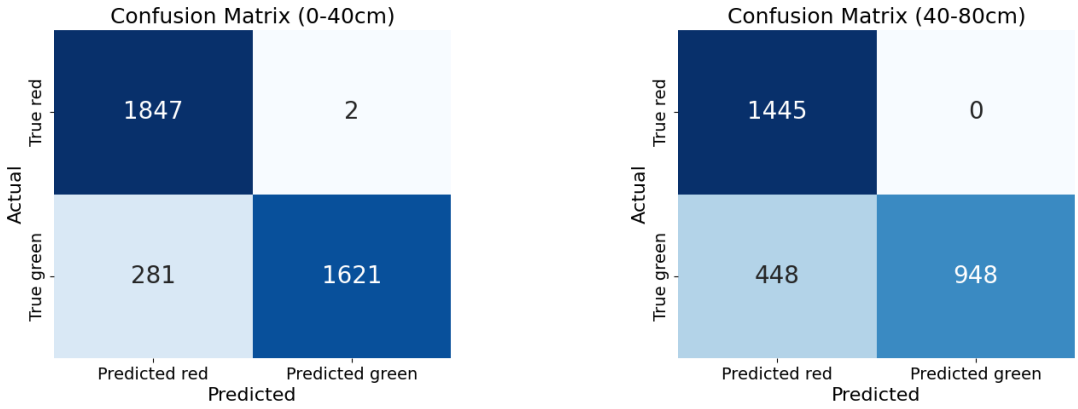
2. Color Classification Accuracy We divide the datasets used for evaluation into eight sets: four directions entering the intersection and two distance ranges per direction, from 0 to 40cm and from 40 to 80cm away from the intersection line (which is 20cm away from the traffic light). The results for each direction and range can be seen in Table 2, along with both confusion matrices in Figure 4.

With these values we can calculate the accuracy of the color classification for each distance range and in total using Equation 3:

- $\mathbf{Acc_{0-40}} = \mathbf{0.92}$,
- $\mathbf{Acc_{40-80}} = \mathbf{0.84}$,
- $\mathbf{Acc} = \mathbf{0.89}$.

		True Red	False Green	True Green	False Red
0-40cm	direction 1	461	0	463	12
	direction 2	463	2	445	8
	direction 3	458	0	439	44
	direction 4	465	0	274	217
40-80cm	direction 1	356	0	320	8
	direction 2	396	0	335	16
	direction 3	359	0	261	53
	direction 4	334	0	32	371

Table 2: True and False Red/Green detections.



((a)) 0-40 cm

((b)) 40-80 cm

Figure 4: Confusion matrices for both 0-40cm (a) and 40-80cm (b) ranges.

The algorithm is effective at detecting red lights but requires improvement in distinguishing green lights, especially from direction 4. Further refinement of the color masking parameters and adjustments for distance-related degradation could enhance overall accuracy, e.g. separate color masks depending on the distance between the Duckiebot and the traffic light. Future work should also focus on mitigating directional biases, such as those observed in direction 4, which is the only direction in which the Duckiebot is directly facing a big light source (the windows of the room), which may cause unexpected glare, making it harder to identify lighter colors, like green.

3. Qualitative analysis The neural network model for traffic light detection performs well overall, with the key limitation being its sensitivity to the distance between the camera and the traffic light. The model worked effectively at moderate distances but showed degradation in performance when the camera was either too close to the traffic light ($\leq$ 20cm) or too far away ($\geq$ 80cm). At very close distances, the model struggled to localize the traffic light accurately, most likely because most (or all) of the traffic light was outside the field of view of the camera. At farther distances, the model had difficulty distinguishing the traffic light from other objects, which could be attributed to the reduced resolution and detail of the traffic light in the image or the lack of images at such distances in the model training dataset.

Another important factor that influenced the detection accuracy was the orientation of the camera relative to the traffic light. The model was most effective when the camera was facing the traffic light directly, but when the camera was angled, especially from certain directions, the model's performance decreased. This is most likely due to the lack of diversity of the training dataset, in which most images were taken perpendicular to the traffic light and on the center of the right lane. Despite these challenges, the neural network was reliable in ensuring that when it detected an object, it was correctly identified as a traffic light, i.e. there were no false positives where the model confused other objects with the traffic light. Additionally, motion blur was not a significant issue at moderate speeds, indicating that the model is robust under typical operating conditions.

As previously mentioned, the color identification component of the system, which relied on color masking techniques, showed promising results but also encountered some challenges, especially when the camera was facing a window with strong light sources. This caused occasional

misidentification of the traffic light color, particularly when trying to distinguish green. Despite this, the detection of red was almost always accurate, regardless of the environmental conditions.

For distances $\geq$ 80cm, the green light detection began to degrade, but the degradation was not severe enough to hinder the system's overall functionality. At close distances ($\leq$ 40cm), there were additional challenges, primarily due to the reflection of the light from the adjacent LEDs on the structure (see Figure 5). However, these issues were not major. As long as the camera's orientation and position were correct, the system was capable of handling these close-distance scenarios adequately.

Figure 5: Example of traffic light image in red with visible reflection from the green adjacent LEDs on the material of the structure.

4 Conclusion

4.1 Conclusion

In this project, we employed machine learning techniques to achieve the detection of traffic light position, size, and color by training and fine-tuning a YOLO model using a real dataset collected from Duckietown. A state machine was designed, and the trained model was ultimately deployed on the Duckiebot. Overall, the performance of the trained model in detecting traffic lights in Duckietown met expectations, achieving an overall accuracy of 0.89, with higher detection accuracy at closer distances. After deploying the code on the Duckiebot, the behavior of the Duckiebot when approaching traffic lights also largely aligned with expectations, confirming the reasonableness of the overall project structure and the effectiveness of the algorithm.

However, certain factors inevitably affected the Duckiebot's behavior. First, outdoor lighting conditions might impact the speed and accuracy of green light detection. Second, lane following exhibited some deviations, occasionally causing the Duckiebot to veer off course, which in turn placed the traffic light outside the camera's detection range. Another issue was the high computational demand of running lane following, traffic light detection, and the state machine simultaneously on the Duckiebot, which limited the ROS system's update frequency, potentially leading to delayed responses.

4.2 Code and Demo

Our code, with a README with full instructions for running and reproducing our results, can be found here, in the final_sm branch:
`https://github.com/ETHZ-DT-Class/DuckietownProject2024/tree/final_sm`.

A demo video showing our robot performing the tasks successfully can be accessed here:
`https://drive.google.com/file/d/1VqIvoUEk8kmqhKwsa1fonQZzSIcDr366/view?usp=share_link`.

5 Next Steps

5.1 Further Work

Integrating proposed state machine:
Given the limitations we faced during implementation (see section 4.1), our first priority would be to complete the implementation of the initial state machine, as depicted in figure 6.

Namely, our first focus would be on red line detection, which could not be integrated previously due to resource constraints. With this module in place, the Duckiebot would be able to achieve consistent stop positions at intersections. This consistency would allow the hardcoded turning maneuvers to reliably position the Duckiebot correctly after each intersection, minimizing the need for subsequent corrections. Additionally, the red line detection module would identify possible turns at intersections by detecting the presence of red lines in each direction, marking valid turns more effectively.

Next, we would focus on fine-tuning the lane-following module or developing a more energy-efficient version. This improvement would ensure that sufficient computational resources are available for other modules, such as red line detection, enabling better performance and integration.

Thirdly, we would revisit the handling of yellow lights, which were excluded from the initial implementation due to frequent misclassification as green lights. Addressing this challenge would enhance the Duckiebot's ability to interact with traffic lights effectively.

Lastly, we aim to implement a robust stopping condition. For instance, the Duckiebot could stop after traveling for a predefined duration or upon completing a certain number of intersection turns. This addition would add a layer of reliability to the system's behavior.

How would we extend this work and what additional skills would be unlocked?
After integrating yellow light handling, a logical extension would be to enable the Duckiebot to interpret standard traffic lights with three stacked LEDs. This enhancement would introduce a broader range of traffic light interactions and extend the Duckiebot's capabilities to more realistic traffic scenarios.

Further extensions could include handling special cases, such as a blinking yellow light that signals an out-of-order traffic light. In such scenarios, the Duckiebot would apply rules like right-of-way, enabling it to navigate more complex, real-world traffic environments.

Additionally, the system could be designed to interact with multiple agents within Duckietown. Since the primary purpose of a traffic light is to manage multiple vehicles at intersections, this would be a critical step in advancing the Duckiebot's functionality. Enhancing inter-agent communication and coordination would significantly improve the system's scalability and realism.

How much effort would be needed?
Implementing these improvements would require significant effort, especially in terms of development, testing, and optimization. Specifically:

1. Completing the red line detection and lane-following modules would involve extensive debugging, fine-tuning, and integration testing.
2. Handling yellow lights and extending capabilities to interpret standard traffic lights would require advanced computer vision techniques, robust classification algorithms, and additional training datasets.
3. Incorporating inter-agent communication would involve designing protocols for coordination, implementing multi-agent simulations, and resolving conflicts at intersections.
4. Handling special cases like blinking yellow lights and right-of-way rules would require both algorithmic development and rigorous testing in varied scenarios.

Overall, while the effort required is substantial, these extensions would significantly enhance the Duckiebot's capabilities, bringing it closer to real-world autonomous navigation systems.

5.2 Considerations

If you had to work on this project from scratch, what would have you changed from the initial plan?
Not much. We believe we took the right approach by starting with a bigger picture and then zooming in on each specific task. This method ensured that when all modules were brought together, the entire implementation made sense and functioned cohesively, like a well-oiled gearbox.

If we were to change anything, it would have been testing the given lane-following module earlier. Its poor performance and high CPU usage were not sufficiently addressed during the development of other modules, which created additional challenges later in the project.

6 Retrospective

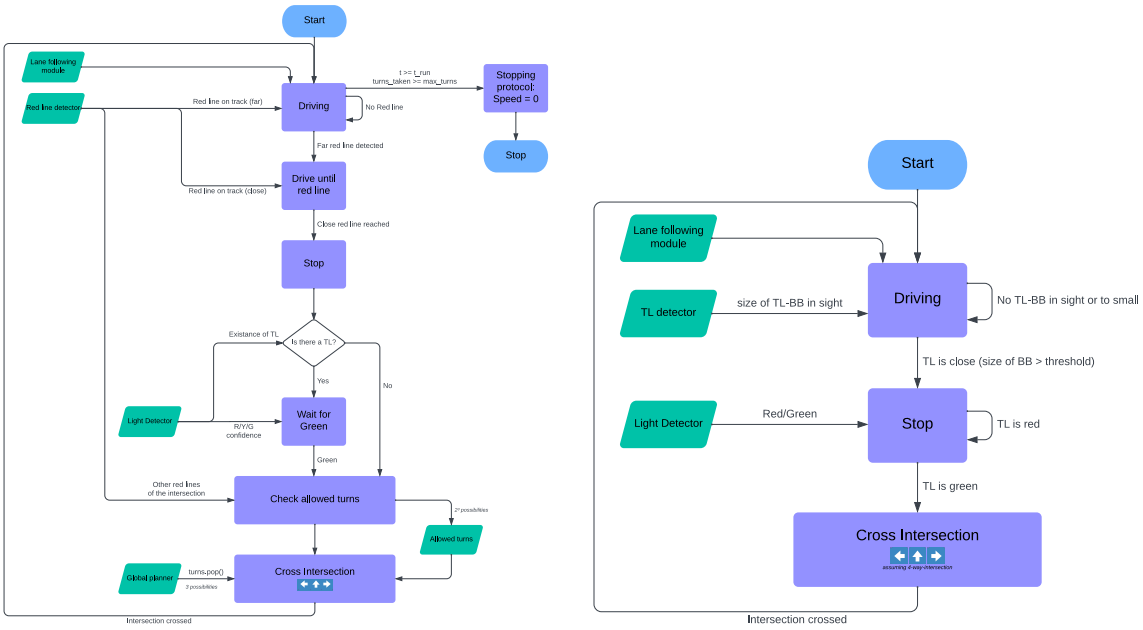
6.1 Project scope

Our project scope has remained largely unchanged. During the initial proposal phase, we divided the project objectives into primary and extended goals. The final project scope aligns with the primary objectives and successfully addresses the core issue outlined in the proposal: traffic light detection. However, due to time constraints and hardware design considerations, we did not pursue the extended goals.

6.2 Approach

Our approach has changed. Initially, we planned to use image processing-based methods. However, in order to enhance stability, we ultimately combined learning-based methods and image processing techniques. The learning-based method was used for detecting the position, while image processing-based methods were employed for color detection.

During the final implementation and deployment of the state machine, we continuously adjusted the design based on real-world conditions. As a result, the final design differed from the initial concept.



((a)) Initial idea of the state machine

((b)) Implemented state machine

Figure 6: Comparison of the initial and implemented state machines.

In addition, our initial idea was to control the Duckiebot to stop by detecting the red stop lines in Duckietown. We wrote code for this, but unfortunately, we didn't have enough time to integrate it with the other components. As a result, we ultimately decided to control the stop behavior by detecting the size of the traffic light's bounding box. When the bounding box reaches a certain size, it indicates that the Duckiebot has reached the intersection and should stop.

6.3 Results

Although more adjustments are needed in real world deployment, overall the final results met our expectations, with the traffic light detection accuracy reaching 0.89. This aligns with our initial commitment of achieving an accuracy greater than 80%, so we would consider it as successful.

6.4 Considerations

My favorite part of this project is the real-world deployment phase. It was fascinating to see the Duckiebot respond to the scenarios in Duckietown according to the design of the code. Through this project, I realized that there is a significant difference between writing code and deploying

it on hardware. I should have prepared for the actual deployment much earlier, gaining a better understanding of the hardware design and making adjustments based on real-world feedback.

References

[1] J. Yang, *Yolo-nano: Object detection model*, GitHub repository, 2024. [Online]. Available: https://github.com/yjh0410/YOLO-Nano (visited on 12/18/2024).

[2] A. Wong, M. Famuori, M. J. Shafiee, F. Li, B. Chwyl, and J. Chung, *Yolo nano: A highly compact you only look once convolutional neural network for object detection*, 2019. arXiv: 1910.01271 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1910.01271.