Duckietown - Safe Navigation

Esra Mehmedova, Ilia Panayotov, Esin Mehmedova
Department of Computer Engineering
Technical University of Munich
Heilbronn, Germany
{esra.mehmedova, ilia.panayotov, esin.mehmedova}@tum.de

1 Abstract

Duckietown is an open-source platform designed for education and research in autonomous robotics and self-driving technologies. It provides a structured, small-scale environment where robotic vehicles, known as Duckiebots, navigate a miniature city using computer vision, machine learning, and control algorithms. These robots are designed to mimic real-world autonomous driving challenges by following lanes, interpreting traffic signs, and avoiding obstacles. Duckietown is widely used for education and research in autonomous systems, providing a hands-on platform for students and researchers to develop and test algorithms for self-driving vehicles.

2 Introduction

In this project, we implemented and refined key functionalities that allow the Duckiebot to operate autonomously while maintaining safety and reliability. These include lane following, which ensures the robot stays within road boundaries, intersection navigation for handling different types of intersections, and traffic sign recognition to interpret road rules. Additionally, we developed an object detection and avoidance system to help the Duckiebot react to obstacles in real-time. By integrating these capabilities, our implementation enables the Duckiebot to navigate complex environments efficiently while adapting to dynamic conditions.

3 Lane Following

The lane following functionality of the Duckiebot is a complex system that involves several components working together to ensure the robot can accurately follow lanes on the road. This system includes the image processing, line detector, ground projection, lane filter, and lane controller. Below is a detailed explanation of how each component contributes to the lane following functionality.

3.1 Image Processing

The image processing pipeline begins by collecting images from the Duckiebot's camera. These images are initially in a compressed format and must be decoded into a usable format for further processing. The next step is rectification, which corrects any distortions caused by the camera lens. This ensures that the images accurately represent the real-world scene. Rectification involves using the camera's intrinsic parameters, such as focal length and optical center, to undistort the images.

3.2 Line Detector

The line detector node is responsible for detecting lane markings (lines) in the images captured by the Duckiebot's camera. These lane markings are white and yellow lines that define the road markings in Duckietown. Additionally, there are red markings used for stopping, which will be discussed later. The lane detector processes incoming images and identifies segments that correspond to these lane markings by performing the following steps:

1) **Image manipulation**: The line detector applies color correction to the incoming images to account for varying lighting conditions. The images are then resized to a standard resolution to ensure consistent processing. Then the top portion of the image is cropped to remove any irrelevant parts that do not include the road.
2) **Color Filtering**: Once the images are acquired, the line detector applies color filtering to isolate the lane markings. The images are converted from the RGB color space to the HSV (Hue, Saturation, Value) colors, which are more suitable for color segmentation. The line detector uses predefined color ranges to filter out the white and yellow lane markings. This step results in binary masks for each color, where the pixels corresponding to the lane markings are highlighted.
3) **Edge Detection**: After color filtering, the line detector applies edge detection to identify the edges of the lane markings. This is done using the Canny edge detection algorithm, which detects edges based on the intensity gradients in the image. The result is a binary image where the edges of the lane markings are highlighted.
4) **Line Segment Extraction**: The line detector then extracts line segments using the Hough Transform algorithm. This algorithm identifies lines in the binary edge image by finding points that align along straight lines. These line segments are then classified based on their color using the color masks generated earlier.
5) **Vector Calculation**: For each detected line segment, the line detector calculates a normal vector. This vector is perpendicular to the line segment and points towards the center of the lane.
6) **Segment List Generation**: The detected line segments, along with their vectors and color classifications, are then packaged into a segment list. This segment list is a structured representation of the detected lane markings and is published for use by other components, such as the lane filter. The segment list includes information about the position, orientation, and color of each detected line segment.

3.3 Ground Projection

The ground projection node transforms the detected lane segments from the image plane (captured by the camera) to the ground plane (the actual road surface). This transformation is essential for accurately estimating the Duckiebot's position and orientation within the lane. This involves the following stages:

1) **Normalization**: The pixel coordinates of the segment endpoints are normalized to a range of [0, 1].
2) **Rectification**: The normalized coordinates are rectified to correct any distortions.
3) **Projection**: The rectified coordinates are projected onto the ground plane using the homography matrix.

3.3.1 Homography Matrix

The homography matrix is a 3x3 transformation matrix that maps points from the image plane to the ground plane. This matrix is obtained through an extrinsic calibration procedure, which involves capturing images of a known calibration pattern (such as a checkerboard) and computing the transformation that aligns the pattern in the image with its known dimensions on the ground.

3.4 Lane Filter

The lane filter is responsible for estimating the Duckiebot's position and orientation within the lane. This estimation is based on the detected lane segments provided by the line detector and processed through several steps:

1) **Prediction**: The lane filter uses the Duckiebot's velocity to predict its new position and orientation. This step involves updating the belief distribution over possible positions and orientations based on the robot's motion model. The belief distribution is represented as a histogram grid, where each cell corresponds to a specific position and orientation.
2) **Update**: The lane filter processes the detected lane segments to refine the belief distribution. Each detected segment provides a "vote" for a particular position and orientation. The filter updates the histogram grid by incorporating these votes, which increases the belief in the cells that correspond to the detected segments. This step helps to correct any errors in the prediction step and improves the accuracy of the estimated position and orientation.
3) **Estimation**: The lane filter extracts the most likely position and orientation from the belief distribution. This is done by finding the cell with the highest belief value.

3.5 Lane Controller

The lane controller is responsible for generating control commands to keep the Duckiebot within the lane. It takes the estimated position and orientation from the lane filter and computes the necessary steering to correct any deviations. The controller uses a proportional-integral-derivative (PID) control algorithm to minimize the lateral and angular errors (d_err and phi_err).

The lane controller computes two main control commands:

- **Linear Velocity (v)**: The forward speed of the Duckiebot. The controller can adjust the speed based on the distance to a detected stop line, ensuring smooth and safe stopping.
- **Angular Velocity (omega)**: The steering angle of the Duckiebot. The controller adjusts the steering to correct any deviations in lateral position and orientation.

These computed control commands are then sent to the Duckiebot's wheels to adjust its motion. The lane controller continuously receives updated lane pose estimates from the lane filter and computes new control commands to maintain the Duckiebot's position within the lane.

4 Intersection Navigation

Navigating intersections is an essential part of our project, as our focus is on ensuring safe navigation within Duckietown. To accomplish this, the Duckiebot must be able to stop at the stop line, detect traffic signs, activate its LED signals, and execute permitted turns. The following sections explain the different processes and components that work together to achieve reliable and safe intersection navigation.

4.1 Stop Line Detection and Stopping

By integrating stop line detection, our robot can reliably identify stop lines and stop at intersections. This process begins with utilizing the segment list from the line detector node, originally used for line following. The system then filters out only the red segments, determines the position of the stop line, and assesses its proximity. If the stop line is in front of the robot, within 0.22 meters, and is located in the lower 20% of the camera image, the robot adjusts its velocity to bring it to a stop. Once stationary, the robot remains idle for three seconds before its velocity is restored to the original value, allowing it to continue driving.

4.2 Sign Detection

Once the robot has stopped at an intersection, it must detect the traffic signs to determine the permitted turns. This is achieved using the Duckietown AprilTags library, which identifies the black-and-white square patterns on each sign and returns a unique ID for each. These IDs are then used to determine the allowed actions for the robot. For this project, we have focused on four types of intersections: T-intersections, 4-way intersections, left turns, and right turns. By recognizing the corresponding traffic signs, the robot can accurately navigate through each intersection type.

4.3 Executing Turns

The Duckiebot uses predefined parameters for each type of turn (left, right, or straight). These parameters include the desired linear and angular velocities, as well as the duration of the turn. The parameters are carefully calibrated to ensure that the Duckiebot can execute the turn smoothly and accurately.

The robot chooses randomly from the available turns based on the traffic signs. For a left turn, the Duckiebot will increase its angular velocity to enable a wide left turn.

For a right turn, it will reduce its angular velocity to be able to make a sharp right turn. For going straight, it will have no angular adjustment. The Duckiebot then executes the turn by maintaining the set velocities for a predefined duration. This duration is carefully chosen to ensure that it completes the turn accurately. During this time, the Duckiebot does not actively monitor its position and orientation but relies on the predefined parameters to complete the turn.

4.4 Turn Signals

Once a turn has been selected, it is important to communicate the Duckiebot's navigation intentions to other robots and ducks within Duckietown. This is accomplished by using its LEDs, which are set to specific color patterns and blinking frequencies for each type of turn. For left and right turns, the corresponding left or right LEDs are set to blink in yellow. If the Duckiebot is continuing straight, the LEDs remain in their normal driving mode, providing clear and consistent communication of its intended direction. This functionality ensures that the Duckiebot can effectively signal its movements, enhancing safety and coordination.

5 Object Detection and Avoidance

A crucial objective of this project is to develop a robust object avoidance system, ensuring that the Duckiebot navigates safely without colliding with other vehicles or pedestrians. To accomplish this, we leverage YOLOv5 to train a neural network capable of accurately detecting robots and ducks. This section outlines the key steps involved in the process, including data collection, model training, and integration into our existing project.

5.1 Data Collection

For data collection, we utilize Roboflow, a computer vision platform designed to assist with dataset management, annotation, and preprocessing for training neural networks. We leverage an existing dataset containing images of ducks and Duckiebots in various orientations to enhance model accuracy. The dataset is strategically divided into three subsets: a portion for training the model, another for validation, and the remainder for testing its performance. This structured approach ensures a well-balanced and robust training process.

5.2 Model training

For training the model, we follow the Duckietown-LX object detection tutorial using Google Colab. The dataset, prepared in the previous step, is uploaded to Google Drive while maintaining the required directory structure. We then utilize the YOLOv5 framework to train the neural network, fine-tuning parameters such as image size, batch size, and the number of epochs to enhance detection accuracy. YOLOv5 is a real-time object detection framework that uses convolutional neural networks to identify and classify objects in images with high accuracy and efficiency. Throughout the training process, key performance metrics, including precision-recall curves and confusion matrices, are generated to assess the model's effectiveness and ensure reliable object detection.

Once the model training is complete and validated, the next step involves uploading the trained model to Duckietown's cloud. This allows us to store and access the trained model for later use in the implementation.

5.3 Implementation

With our training now completed, the Duckiebot can detect objects using its camera. The system provides bounding boxes that outline the detected objects, class labels indicating whether the object is a Duckiebot or a duck, and confidence scores that represent the model's certainty in its classification. The system then classifies the detected objects and applies a decision-making process to determine whether the Duckiebot should stop or proceed. It evaluates factors such as the confidence score and the size of the bounding box, to assess the proximity of the object. Additionally, the object's position relative to the center is considered to avoid unnecessary stops for objects outside the main path. If a close object, such as a Duckiebot or duck, is detected within a critical central region, the Duckiebot will stop. Otherwise, it will continue moving at a set velocity, ensuring smooth navigation while avoiding collisions.

6 Conclusion

In this project, we successfully implemented and integrated key functionalities that enable the Duckiebot to navigate autonomously and safely within the Duckietown environment. Through the development of lane following, intersection navigation, traffic sign recognition, and object detection and avoidance systems, we created a robust platform capable of handling real-world autonomous driving challenges on a smaller scale.

The lane following system leverages a combination of image processing, line detection, ground projection, and lane filtering to ensure the Duckiebot maintains accurate lane positioning. Intersection navigation integrates stop line detection, traffic sign recognition using AprilTags, turn execution with predefined motion parameters, and LED signaling to communicate navigation intentions clearly. Additionally, the object detection and avoidance system, powered by YOLOv5, enhances the Duckiebot's ability to recognize and respond to dynamic obstacles, ensuring smooth and collision-free operation. Together, these integrated systems enable the Duckiebot to operate autonomously in a dynamic and complex environment, closely mirroring the challenges faced by full-scale autonomous vehicles.

References

[1] https://github.com/duckietown/dt-core

[2] https://github.com/duckietown/lib-dt-apriltags

[3] https://samuelfneumann.github.io/posts/duckie_4/

[4] https://github.com/duckietown/lx-object-detection

[5] https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data

[6] https://colab.research.google.com/github/duckietown/duckietown-lx-recipes/blob/mooc2022/object-detection/assets/colab/dt_object_detection_training.ipynb

[7] https://universe.roboflow.com/yellowfever/duck-dmrw1

[8] https://universe.roboflow.com/duckietown-rvbyd/duckiebots