Dynamic Obstacle Avoidance
(proj-lf-vop)

Nikolaj Witting, Fidel Esquivel Estay, Johannes Lienhart, Paula Wulkop

December 2019

1 The final results [1]

The final result of this project is shown in the video found at: `https://vimeo.com/380189802`

2 Mission and Scope

Our mission is to develop a feature that will enable Duckiebots to detect and pass dynamic or static obstacles such as a slowly moving Duckiebot, stopped Duckiebot or a static Duckie, while being aware of oncoming traffic.

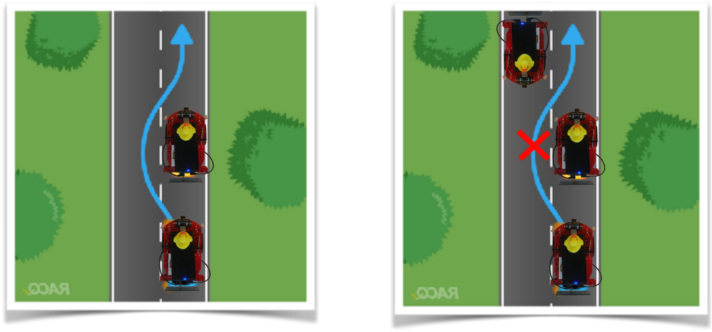

Figure 1: Illustration of our mission. On the left a slowly moving or static Duckiebot is overtaken, while on the right the rear Duckiebot is staying behind the obstacle to avoid oncoming traffic.

2.1 Motivation

In general Duckiebots are able to drive around a Duckietown by themselves based on a lane-following function. If another Duckiebot is blocking the road

the first mentioned Duckiebot will be able to stop to prevent a collision, but it will be stuck behind the slow or stopped Duckiebot. This a serious challenge for Duckietowns with multiple Duckiebots; a failure for a single Duckiebot on the lane will quickly create a traffic jam. A smooth flow of traffic in Duckietown is very important, but so is pedestrian safety. Therefore Duckies should also be detected and passed without imposing any danger.

2.2 Existing solution

There is no existing solution for passing obstacles within the Duckietown pipeline. The current implementation is able to detect the back bumper plate of Duckiebots and prevent a collision by keeping a safe distance.
Work from previous years looked into static obstacle avoidance of Duckies. Time was spent looking into these solutions but we were not able to verify nor use their solution.

2.3 Opportunity

The baseline Duckiebot detection is based on pattern recognition of the bumper plate. This only enables detection of Duckiebots from the back, and not the front. It is necessary to be able to detect Duckiebots from the front as well as we need to pass into the other lane. Therefore a new detection method has to be developed for the detection of a Duckiebot from the front. Introducing the *HoodPlate*^{TM}, a bumper plate for the front of your Duckiebot. Jokes aside, instead of altering the current hardware, this provides an opportunity for developing a new method, which will be able to detect Duckiebots from both the front and the back using computer vision and LED detection.
This LED detection will be based on OpenCV functions for blob detection and the pinhole camera model to estimate the positions of the detected Duckiebots.

2.4 Preliminaries

Familiarity with the pinhole camera model as well as knowledge about blob detection is necessary for our detection method. Furthermore a general understanding of the Duckietown pipeline, specifically the functionality around lane-following is necessary.

3 Definition of the problem

For this project the final objective is to:

- Enable a Duckiebot to safely pass a slow or static Duckiebot and static Duckies while driving in a simple Duckietown environment.

To achieve this some intermediate objectives can be defined as:

- Enable detection of Duckiebots from the front and back

- Enable detection of Duckies
- Being able to drive a Duckiebot into the left lane and back again while performing lane-following

Assumptions made for limiting the scope:

- Only straight lanes without turns are considered
- Obstacle Duckiebots will not weave across lanes
- Obstacles are confined within the lanes

To quantitatively test our solution, the fraction of successful passes of the obstacles will be measured for the defined use case.

4 Contribution / Added functionality

The contribution of our project can be split into the three parts: "Duckiebot detection", "Duckie detection" and "Decision logic and passing maneuver".

4.1 Duckiebot detection

Detection of Duckiebots and their relative position is essential for performing a safe overtake. Our detection algorithm is be based on LED detection. Here two features can be exploited, first, the LEDs should be easily detectable with a high brightness value. Second, the LEDs in the front and in the back have the same fixed spacing between them, which can be used for estimating the relative position.
First the LED key points need to be extracted. Both the white and the red LEDs are very bright with RGB values close to $(255, 255, 255)$. The image is converted to a grey-scale image and threshold mask can be applied at a high value of 235 which will keep the LED features and other bright parts of the image, this can be seen on figure 2.

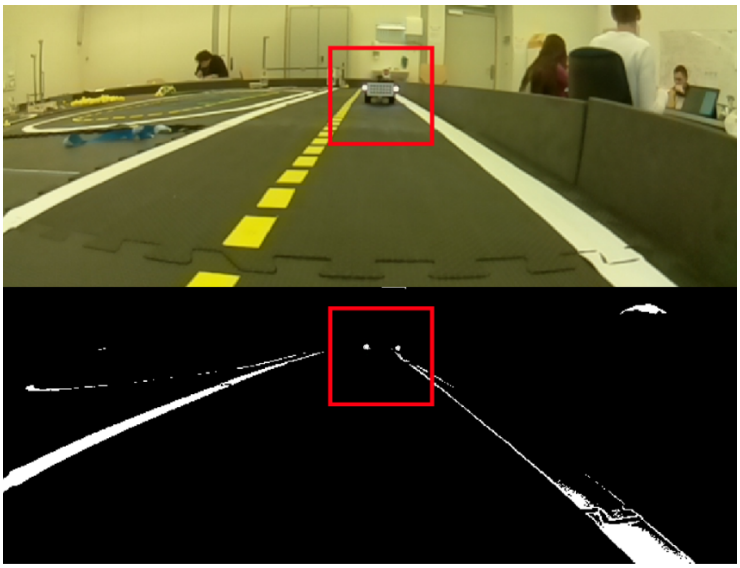

Figure 2: The cropped input image seen on top, with the thresholded binary image below.

To find the LED features a blob detector is used to extract circular blobs in the image. The LEDs are perfectly circular, and most of the noise is caused by the white lines which are not circular. The blob detector is implemented with OpenCV and tuned based on size and circularity, yielding an image with the LEDs detected and typically a handful of false positives.
The Duckiebot images are quite distorted radially and should be undistorted such that the coming calculations based on the pinhole model are accurate. This undistortion is not done for the entire image as this is very resource demanding, but instead we have implemented an undistortion function that just works on the feature key points using the camera calibration data.
It is then necessary to find the undistorted key points corresponding to the actual LEDs. This is done based on the assumption that Duckiebots are always perceived straight on and the fact that the size of the LED blobs should be similar. So each key point is compared to other key points and checked whether they have similar size and have similar y-coordinates, if this is true, we determine the matching key points as an LED pair.
Finally, we need to determine whether the LED pair is red or white. This creates some challenges because of saturation of the image sensor. To the human eye the LED looks clearly white or red, but for the camera both cases look very white as seen on figure 3.

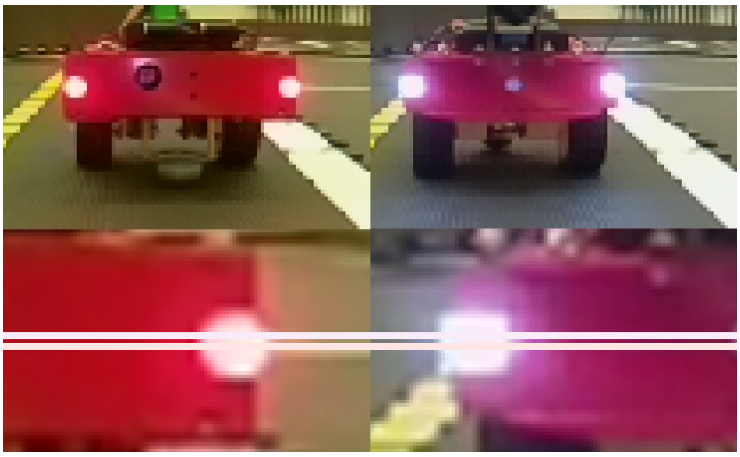

Figure 3: The similarity between the red and white LEDs as percieved by the Duckiebot is illustrated here. On the left are the red LED and on the right is the white LED. In the bottom half, the center color of each LED is drawn through itself and the opposite LED to show the similarity between them.

To determine the LED color we look at the RGB value at the keypoint position. We have looked at several red and white LEDs under different illumination conditions and found typical intervals for the different colors as seen in table 1. This shows that only the blue color channel have a distinct, although still subtle, difference between the white and red LEDs. We therefore apply a threshold filter to this channel to determine the LED color. Since a red LED will trigger an overtaking sequence and a white LED will trigger a safety stop, we determine that it is better to think a red LED is white than vice versa and use a threshold value of 225 to separate the groups that are not disjoint.

LED\color	R	G	B
white	∼255	245-255	230-255
red	∼255	240-250	215-235

Table 1: Experimentally found RGB values for red and white LEDs

With the found LED pairs the relative position of the opposing Duckiebot can then be calculated using the pinhole camera model. Here the following relation can be used:

$$\frac{x}{f} = \frac{X}{Z}$$

The focal length f is known, the real LED pair spacing X is measured to be $0.12m$, and the image spacing between the LED key points x is found based on the key point positions. Then the depth Z is readily calculated. The relation is also illustrated in figure 4.

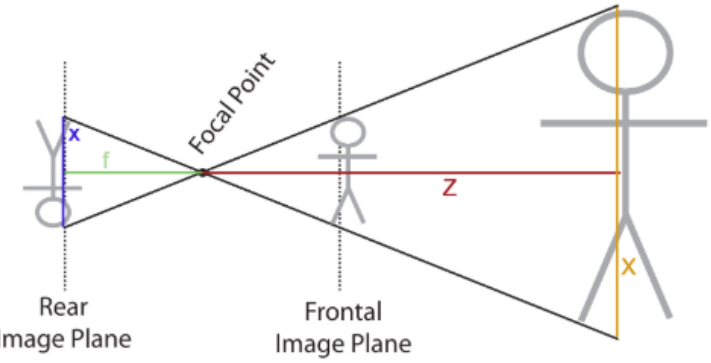

Figure 4: The cropped input image seen on top, with the thresholded binary image below.

The same procedure is used to find the relative position offset in the left-right horizontal direction. With this we get a pretty accurate position estimate of the opposing Duckiebot, which is used later in the decision logic.

4.2 Duckie detection

In this section it will be described how standard yellow Duckies are detected and how their location is computed. For the detection, the Duckie's most dominant feature is used: its yellow color. As a first step, the image is cropped horizontally so that it contains only the road. Then a yellow mask is applied to the HSV image. This results in an image as shown in figure 5.

Figure 5: Yellow mask of an image with Duckie on lane.

The challenge is now to differentiate between Duckies and yellow line segments. For this, the inertia of the two objects is used. While the line segments are elongated and therefore have a high inertia, the duckies are compact and have a low inertia. An OpenCV blob detector is used to detect the blobs on the mask with low inertia. This gives good results as long as the road is straight and the Duckiebot is driving in the middle of the lane. In curves and in oscillating lane following, some line segments are viewed from different angles and therefore

wrongly classified as Duckies. To prevent these false positives, a tracking system was implemented. This detects a Duckie only if it has been detected within a certain range already in the previous frame. This method significantly decreased the false positive rate, but some false positives might still occur.

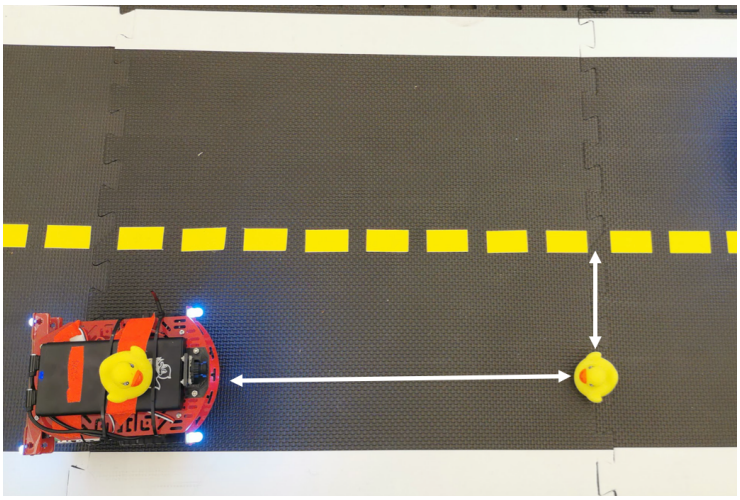

Figure 6: Duckiebot driving towards Duckie

As a next step, it is determined if the detected Duckie is blocking the road and if yes, on which lane. For this, the location of the Duckie is computed with the pixel2ground projection. Using the lane_pose information from the lane_filter, the distance from the Duckie to the middle line is calculated. From this distance it can be inferred if the Duckie is on the right or on the left lane. Of course this works only on straight tracks and with a well functioning lane_filter. Additionally, the distance from the Duckiebot to the Duckie is computed. Finally the information about the location of all detected Duckies and on which lane they are is published to the dynamic_controller_node, which is later used in the decision making for the passing maneuver.

4.3 Decision logic and passing maneuver

In this section the decision logic, which decides whether the Duckiebot overtakes or not, is described. Also, the passing maneuver is explained. Both parts are implemented and executed within our dynamic_controller_node which subscribes to the led_detection_node and the duckie_detection_node in order to get the information where the detected Duckiebots and detected Duckies are, respectively.

Decision logic

The decision logic checks certain criteria and decides if the overtaking is executed. Figure 7 represents this logic and points out when the overtaking ma-

neuver is executed. In particular the logic checks for the following cases:

- Whether there is an obstacle on the right side of the road
- Whether the detected obstacle is close enough for overtaking
- Whether the left lane is free to perform the overtaking maneuver

If the obstacle is too close it does a emergency stop until the obstacle is removed. During the overtaking the logic also checks whether there is an obstacle on the left lane what will result in an emergency stop until the obstacle is removed. When the obstacle gets removed the Duckiebot will simply go back to the right lane. Here, it is assumed that a person removes the obstacle and also checks if the Duckiebot can go back to the right lane.

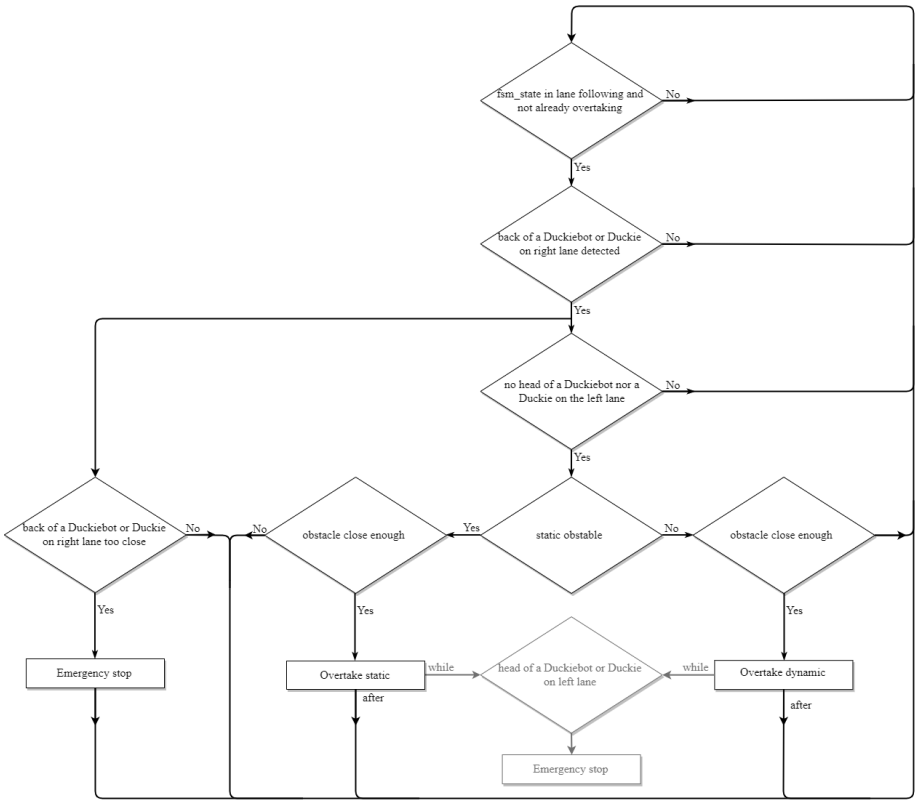

Figure 7: Decision logic

Passing maneuver

The overtaking in our project is achieved by altering the d_offset parameter of the lane_controller_node. This parameter gets used by the lane_controller

(active during lane following) and represents the desired offset of the Duckiebot to the middle of the right lane. In order to determine the optimal overtaking maneuver, we tested the step response of the d_offset for a single overtaking action. The results for this can be seen in 8. From this data, it can be seen that using the d_offset is a reliable method to change the relative lane position, however, it still presents a high levels of noise when in the left lane.

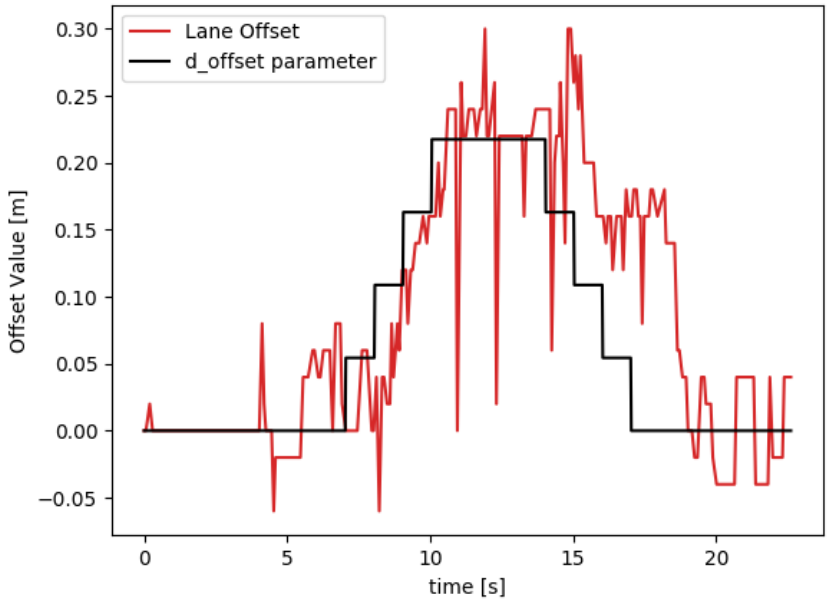

Figure 8: Lane offset response to linearly changed d_offset parameter

The shape of our overtaking maneuver is shown in figure 9. When the Duckiebot is on the left lane and has a large angle ϕ ($\sim > 45°$) we observed that the pose belief is unstable and jumps substantially to the right lane (possibly because the Duckiebot does not see enough of the lane to give a solid estimate), followed by an according reaction from the lane_controller which in our case resulted in a Duckiebot going off-road. Therefore, the shape for the transition from one lane to the other was chosen sinusoidal to avoid this situation.

The overtaking is divided in three phases, the first one is the transition phase from right to left lane, the second one the overtaking phase and the third phase is the transition back to the right lane. In the first and third phase the d_offset parameter gets changed in discretized sinusoidal steps. The transition time is 4 seconds what is an empirically found value which resulted in a stable transition. The time which is spend on the left lane is also constant, but here two values

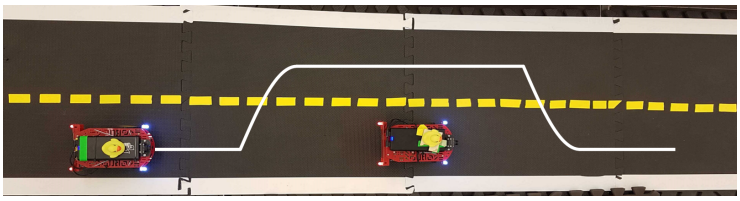

Figure 9: Overtaking maneuver

are used, one for static obstacles (2 seconds) and one for moving (4 seconds). Originally the time spend on the left lane was designed to be variable dependent on the relative velocity between obstacle and overtaking Duckiebot, but after testing we figured out that our relative velocity estimation of Duckiebots was not reliable enough in order to be used. Also, an increase of the speed during driving on the left lane was implemented in order to minimize the overtaking time, but since the lane following could not cope with the increased speed in most cases this was not used in the final version.

5 Formal performance evaluation / Results

Overall the goal of this project was reached: A Duckiebot can safely detect and overtake Duckies, static Duckiebots as well as slowly moving Duckiebots. However, there are still a few reasons why the maneuver can fail from time to time. For once, the detection of the Duckies gives sometimes false positives when it mistakes a line segment for a Duckie. The Duckiebot detection sometimes detects false positives if there are too many other light sources around, e.g. from other Duckiebots or light reflections on shiny objects. During overtaking and when driving on the left lane, the lane following can get unstable. As for the emergency stop, it sometimes detects an obstacle, correctly stops, but then starts driving again if it fails to re-detect the object. Despite all these possibilities of failure, the performance was quite good, as it can be see in table 2.

	Detected	Overtake successful	Emergency stop while overtaking
Static Duckiebot	12/12	10/12	8/12
Duckie	20/20	15/20	10/12
Moving Duckiebot	13/13	12/13	–

Table 2: Performance

The obstacle detection worked correctly in all test cases. However, in some cases especially for the Duckies, the obstacles were detected too late. The Duckiebot started to overtake and then crashed into the obstacle because it was

too close already. Overall an overtake success rate of 83% for static Duckiebots, 75% for Duckies and 92% for moving Duckiebots was achieved.

6 Future avenues of development

The dynamic obstacle avoidance functionality was shown to work for all our intended scenarios, given the assumptions we made hold true. However, the node's behaviour is yet far from optimal, leaving room for further development. The main key areas that we identified for future work are the performance of the lane offset estimator, the implementation of a reliable velocity estimation, and a calibration model for lane following at different speeds.

Lane offset estimation
One of the major challenges while building on the existing Duckietown Lane Following pipeline related to the implementation of the lane offset and pose estimation. These functionalities have shown a large degree of repeatability for straight and curve lane following, however, we noticed that at edge cases, such as when changing lanes, or having a large lane pose ϕ ($\sim > 45°$), the estimates were less accurate and affected the behavior of our controllers. Possible improvements for these edge case scenarios have already been explored by other teams, such as the Machine Learning based lane following developed by the proj-lf-ml team 2019 [2]. Therefore, a better lane offset estimation has the potential to achieve a more repeatable performance for the dynamic obstacle avoidance task.

Duckiebot velocity estimation
Estimating the velocity of a moving Duckiebot is an essential part of the ideal dynamic obstacle avoidance, as it directly determines whether an obstacle can be overtaken, or not. In our current implementation, we assume that if the distance from a vehicle in front goes under the overtaking threshold, then the vehicle is an adequate obstacle for overtaking. This, however, might not always be the case. A clear counterexample is that a slow moving Duckiebot in front might still be moving too fast for overtaking, therefore, speed adjustment would be desired instead.

The current Duckiebot detection is implemented using the blob detection and pinhole camera model, as detailed in section 4.1. This method was shown to work for our intended purpose of detecting and extracting position data for the Duckiebots in our field of view, which was used for triggering the overtaking maneuver if an obstacle was detected. However, when attempted to obtain the Duckiebot's velocity through the finite difference of this position signal, it proved ineffective. The position data, although accurate within a given range, fluctuated significantly preventing an adequate velocity measurement. Future work might include the development of a more reliable LED perception and position estimate, which will result in higher confidence velocity estimates. This can be done by implementing tracking of detected Duckiebots, preventing noisy signals from being identified as moving vehicles.

Another possibility is to use other estimation mechanisms such as a linear quadratic estimator for the position and velocity signals. This method would help to develop a more reliable velocity estimation, without needing to modify the perception pipeline.

Variable speed lane following
When overtaking a moving or non-moving obstacle, it is oftentimes desired to adjust the overtaking speed. As part of the Dynamic Obstacle Avoidance project, we looked into having a variable speed control command, adjusting to different speed ranges for moving obstacles. Besides the aforementioned difficulty estimating the obstacle's speed as mentioned in the previous section, a major challenge with this solution was the instability of the lane following pipeline when the speed was dynamically adjusted. When running this approach, most times it was seen that the lane following quickly turned unstable veering off the lane. We implemented this by changing the v_bar parameter in the lane_controller_node, but were unable to obtain a working solution that could be used for the overtaking maneuvers. A second approach was to modify the gain value for the kinematics_node, but also failed when implemented dynamically. Therefore, a future development opportunity is to improve the stability of the lane following when changing speeds. This can be done by modifying the lane control model to change the wheel trim values depending on the desired speed. Some work has already been done in this area, therefore, integrating it with the current lane following solution should be possible.

References

[1] proj-lf-vop 2019. Dynamic obstacle avoidance. Available at `https://github.com/duckietown-ethz/proj-lf-vop`.

[2] proj-lf-ml 2019. Machine learning based lane following. Available at `https://github.com/duckietown-ethz/proj-lfi-ml`.