MadDuck Avoidance

Intelligent Robotics - M.EIC - FEUP - 2022-2023

1st João Pires
M.EIC
FEUP
Porto, Portugal
up201806079@up.pt

2nd Julián Herrero
M.EIC
FEUP
Porto, Portugal
up202202340@up.pt

3rd Lisa Sonck
M.EIC
FEUP
Porto, Portugal
up202202272@up.pt

***Abstract*—MadDuck Avoidance project that includes the detection of objects - rubber duckies - using an annotated dataset and the algorithm Yolov3, the object avoidance and all the procedures, implementation and conclusions taken. Development made entirely using the DuckieTown Simulator and Python as the main programming language.**

***Index Terms*—duckietown, machine learning, computer vision, object detection**

I. Introduction

This paper describes in detail the procedure and a concrete implementation of the MadDuck Avoidance problem. This problem divides in two main parts: one related with the object detection and the other one with the object avoidance.

This document starts with an introduction to the DuckieRobot and to the DuckieTown Simulator. Then, there's a brief explanation on the basic movement of the DuckieRobot, followed by the two main topics, object detection and object avoidance. Finally, the authors of this paper conclude and acknowledge on the work made.

II. DuckieRobot and DuckieTown Simulator

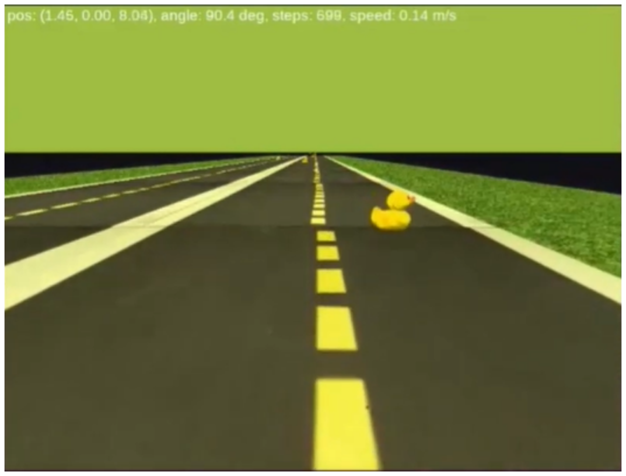
Despite having a physical DuckieRobot, the current status of it didn't allowed to be used for test purposes, so the authors of this paper decided to go with the Gym-Duckietown [5], a Python/OpenGL (Pyglet) simulator for DuckieTown.

Despite the simulator having already predefined environments and maps, the authors of this paper decided to create a custom map, just using the 'straight/N' and 'floor' tiles available, as well as placing duckies to serve as objects to be avoided.

III. DuckieRobot Basic Movement

The DuckieRobot has the possibility to be controlled directly by using the arrows of the keyboard or by moving automatically by pressing the 'Space' key, which will make to DuckieRobot go in straight line, until it detects an object to avoid, entering then in the avoidance procedure.

The movement is controlled, in both ways, by changing the linear and angular velocities of the robot.



```
pos: (1.45, 0.00, 8.04), angle: 90.4 deg, steps: 699, speed: 0.14 m/s
```

Fig. 1. Screenshot from the custom map on the DuckieTown Simulator.

IV. Object Detection - Rubber Duckies

A. Dataset

The first and foremost part of object detection is having an adequate and useful dataset. This dataset should exist out of many pictures, which contain the object in various ways. Besides this, also a good annotation of the object in these pictures is necessary. Building an own dataset is very time consuming and often not as good annotated, since as an individual the probability of making mistakes is higher. Using a publicly available and already annotated dataset is therefore a preferred option. Various users already have used this dataset and relevant feedback was implemented.

For this project, the dataset, made by Soroush Saryazdi and Dhaivat Bhatt [1], was used. This dataset consists out of 1956 images in which 3 different objects are annotated; cones, other duckie robots and rubber ducks. In total, 5068 objects are annotated in these images. Because for this project only rubber ducks needed to be detected, the dataset was filtered, with only keeping the images containing a rubber duck and the annotations corresponding to the rubber ducks. This resulted in a dataset of 944 images. In Fig. 2 an example of an image in the dataset is given, with and without annotations. The red box on the right image represents the annotations made by Saryazdi and Bhatt.

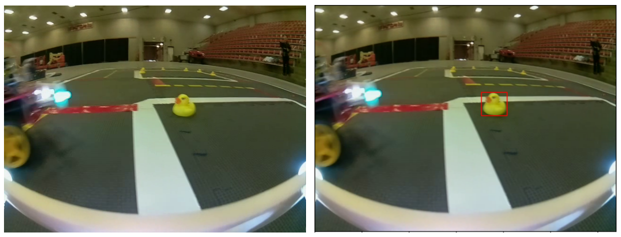


Fig. 2. Example of an image in the dataset, without and with annotation.

B. *Yolov3*

For the object detection, the algorithm You Only Look Once Version 3 (Yolov3) is used [2]. This algorithm is a real-time object detection algorithm that can identify specific objects in images, videos and real-time feeds. Its prediction is based on a convolutional layer that uses 1×1 convolutions, and, compared to other detectors, this algorithm is extremely fast and accurate [3].

1) Preparation: Yolov3 uses a specific annotation format, as seen in Eq. 1. The class indicates which object is annotated, for this project this is always a rubber duck (duckie). $< x >$ and $< y >$ are the center coordinates of the bounding box and $< width >$ and $< height >$ refer to the box' dimensions. All these values are made relative to the size of the image, which means that to have the exact center coordinate of x, you have to multiply the $< x >$ value with the width of the image. This gives the advantage that even with resizing the image the box will stay at the right place.

$$< class >< x >< y >< width >< height > \tag{1}$$

The annotations of the dataset of Saryazdi and Bhatt are in a different format, namely $< x >< y >< width >< height >$ where $< x >$ and $< y >$ are the exact coordinates of the top left corner. These values are not relative to the dimensions of the image, so easily resizing of the image is not possible. The $< width >$ and $< height >$ are representing the dimensions of the bounding box in pixels.
Therefore, to be able to use this dataset with Yolov3, the annotations needed to be converted.
Besides this, the available images were split into a training and test set. The training set consists out of 630 images and the test set out of 315.

2) Implementation: After the annotations were converted and the different sets were made, Yolov3 could be implemented. For this, the standard Yolov3 model was altered to the specific needs for this project. The number of subdivisions was put to 64 and the width and height both to 416. These values were chosen, since training the Yolov3 requires a lot of memory and power from the computer and a smaller batch size or bigger dimensions resulted in memory errors.
The maximum amount of batches was set to 4000 and the steps to 3200 and 3600. For each yolo layer in the network, the class variable was set to 1, since only duckies need to be recognized, and for the convolutional layer above the yolo layer the number of filters is set to 18.
The model was then trained with darknet against the image training set and with the pretrained convolutional weights "darknet53.conv.74".

3) Results: The training took around 5 days to complete. The training curve can be seen in Fig. 3. At around 200 iterations the loss starts to decrease and at around iteration 1200 the loss is already less than 0.5. The average loss at 4000 iterations was equal to 0.1476. After validating with the test set, the obtained average precision is 90%.

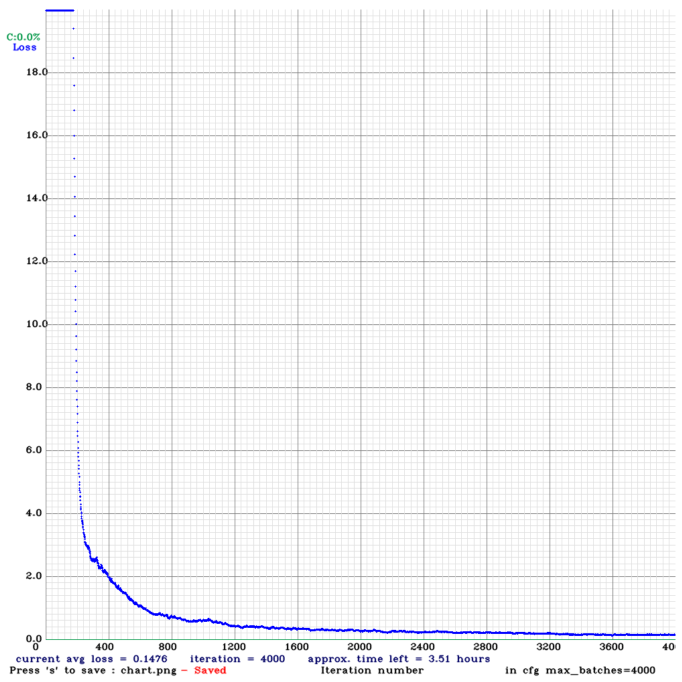


Fig. 3. The training curve.

V. Object Avoidance

Having the object detection part concluded, the following one refers to the avoidance of the objects. In order to test an implementation of it, the custom map already described was used.

Every 10 frames, the current frame is being analysed to see if it detects a duckie. If so, inside the program, the position (x and y coordinates) of the duckie and it's width and height are available to support the decision on what to do next.

Before moving from the straight forward movement to the curvature, the code verification checks whether or not the duckie is in the trajectory of the DuckieRobot by comparing its position (the coordinates of the box surrounding the duckie are used) with a 'border threshold' (200 for the left and 440 for the right).

Then, it keeps checking how close the duckie is and, only once it's close enough, it starts the curvature around the duckie, in order to avoid it. By close enough, the authors of the paper mean that the box that surrounds the detected duckie has a height equal or above to 40. This follows the principle that the closest to the duckie, the higher will be the box surrounding it.

The curvature takes into account the position of the duckie to avoid. If the duckie is mostly to the right of the DuckieRobot, the curvature will be made by the left of the duckie. Otherwise, by its right. In order to detect if the duckie is mostly to the right or left, the field of view was divided in half and the position of the box surrounding the duckie is compared against that value (320).

The curvature itself is achieved in 4 steps, with the first one being equal to the last and the second equal to the third. The first and last movements are a curvature to the left, during 50 frames each, while the second and third are a curvature to the right, during 50 frames each. The curvature is made with an angular velocity of 1.2 (in absolute terms) and a linear velocity of 0.2, with this last one being the same as the one used for the straight line movement.

After the curvature, the DuckieRobot gets back to the straight line movement and keeps that movement until it finds another duckie, repeating then the curvature procedure described above.

VI. Conclusion

The results obtained both in the object detection and avoidance show how good both major parts were achieved.

By completing both parts with success, it was proven that it's possible to program the DuckieRobot to detect and avoid objects, despite the limitations described next.

By using the simulator instead of the physical robot, the development cycle was faster, but with significant differences from the code that would need to be developed in case the authors of the paper were working with the physical robot.

Acknowledgment

With this project, the authors of this paper were able to apply an algorithm (Yolov3) with an annotated dataset to the object detection of duckies in the context of the DuckieTown, implementing then the logic to avoid the duckies.

The object detection part was the one that got most of the attention and effort. However, the authors of the paper consider this project as a complete and concrete example and implementation of the MadDuck Avoidance problem.

There are, however, limitations to the implementation that was made, which represent the future work to be made in order to make this project even more complete. The main one is the fact that the DuckieTown robot is only able to move forward and avoid obstacles, so one major improvement would be to make it move freely by any map, detecting its limits and the road itself.

Another limitation is regarding multiple side-by-side duckies avoidance, since, as of now, only the avoidance of one duckie at a time is possible.

Last, but not least, as already mentioned, this project was conducted using the DuckieTown Simulator, which was good to prove the concept, but the usage of the physical robot is recognized also as future work.

References

[1] S. Saryazdi, D.Bhatt. ”The Duckietown Object Detection Dataset”. Github.com. https://github.com/saryazdi/Duckietown-Object-Detection-LFV/blob/master/DuckietownObjectDetectionDataset.md.

[2] V. Meel. ”YOLOv3: Real-Time Object Detection Algorithm (Guide)”. viso.ai. https://viso.ai/deep-learning/yolov3-overview/

[3] J. Redmon. ”YOLO: Real-Time Object Detection”. pjreddie.com. https://pjreddie.com/darknet/yolo/.

[4] DuckieTown. ”DuckieTown Simulator Maps”. github.com. https://github.com/duckietown/gym-duckietown/tree/master/gym_duckietown/maps.

[5] DuckieTown. ”DuckieTown Simulator”. docs.duckietown.org. https://docs.duckietown.org/daffy/AIDO/draft/dt_simulator.html.