Object Identification and Displacement Computation

RIS Project - Duckietown
Prof. Dr. Alanwar Abdelhafez Amr

Authors:
Assylbek Sakenov
Denisa Checiu
Kuranage Roche Rayan Ranasinghe

1. Introduction

1.1 Duckietown

The RIS Project class for which we created this project is part of the Duckietown project, which started at MIT in 2016. The main elements of this project, which encourages learning and research in the autonomous robotics domain, are duckiebots. To gain a better understanding, duckietowns are the urban environments: roads, constructed from exercise mats and tape, and the signage that the robots use to navigate around, while duckiebots are low-cost mobile robots that are built almost entirely from off-the-shelf parts.

Figure 1.1.1 : Picture of a duckiebot

Figure 1.1.2 : Picture of a duckietown

1.2 Object Detection

Object Detection is a task related to computer vision and image processing that aims to find objects of certain classes(humans, items, animals, etc.) in images or videos. There are various methods and algorithms that can range from Colour Thresholding to Convolutional Neural Network models that involve Deep Learning based on datasets with 1000, 3000, or even 10000 images, depending on the number of classes and other configurations. Each one of them has its own advantages and disadvantages, and of course, its best use.

1.3 Displacement computation

Determining the distance from a camera to a beacon, marker, and, or object is a well-studied problem in the computer vision/image processing area. There are several techniques to perform it.

2. Related work

Object Detection and Distance calculation have many applications, for example, object detection: Obstacle Avoidance, Human Counting, and Presence Check. For Distance calculation: Mapping, Motion Planning, and Trajectory Prediction. As it can be observed there are many instances for both technologies.

Let Obstacle Avoidance be our main sample for analysis. Obstacle Avoidance is a crucial part of any automotive machine. Because depending on the environment, the absence of any safety measures may lead to a crash or even human casualties. That's why it is important to get the exact distance toward a certain object like a car or a human. Using this data machine or onboard PC can take action and prevent a potential accident.

3. Problem Formulation

As the duckietown project can be seen as a simulation of an autonomous or manually-controlled car driving through a city, our idea was to conduct object detection (of duckies & duckiebots) and displacement computation to each detected object, which can further be considered an obstacle or a stopping point.

Thus, we are striving to create practical real-time object recognition algorithms, through which the robot can recognize the object in its own camera view at each frame time with the lowest delay and highest accuracy. By using a convolutional neural network (CNN) through the YOLOv5 software (you only look once), we conducted object recognition on a Raspberry-Pi-based mobile robot linked with ROS(Robot Operating System) - namely our duckiebot, and concluded that it shows high performance on real-time object recognition of static or dynamic objects.

4. Approach

We started off by setting up and calibrating our robot, the duckiebot, by following the steps specified in the manual. Our approach will be explained thoroughly in the next two sections, which separate the main two parts of our project: object detection and displacement computation.

4.1 Object Detection

4.1.1 Preparing the training dataset

The first step was to prepare the training data. We accessed the dataset provided in the official duckietown repositories, which contains images from logs of a duckiebot in a duckietown in a variety of lighting conditions with objects.

Figure 4.1.1 : Images from the official duckietown dataset

Furthermore, we used the VoTT software (Visual Object Tagging Tool) to annotate the images. In each image, we manually labelled each object type we later want to detect, in our case duckies and duckiebots, using boxes to delimitate the margins. Below, we attached a screenshot which better showcased our process.

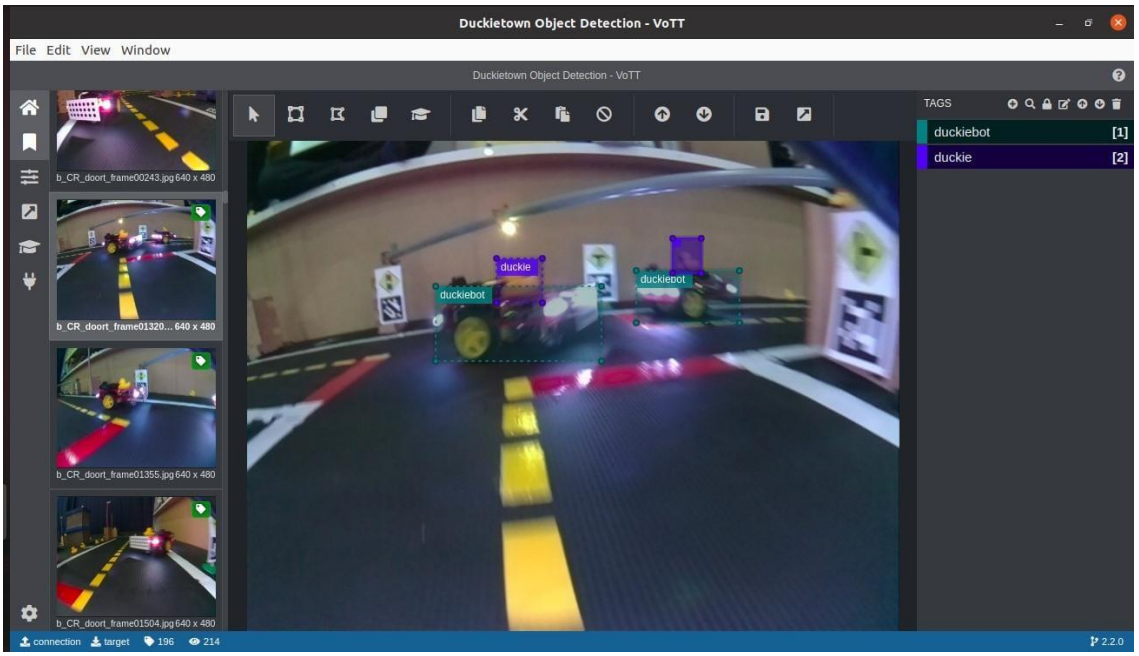

Figure 4.1.2 : Using the VoTT software to annotate our images

After gathering the annotated images, we used the Roboflow software to perform data collection and generate our final data set. In order to increase the accuracy of the algorithm, we generated a bigger dataset. We augmented our dataset by blurring out or zooming in on the already annotated images, thus creating new training data. Moreover, the blurring prepares for different camera focus or quality, while zoom in/out prepares for different angles or proximity to detected objects, both of these processes helping our detection algorithm be well-rounded to changes in the environment or the to-be-detected objects.

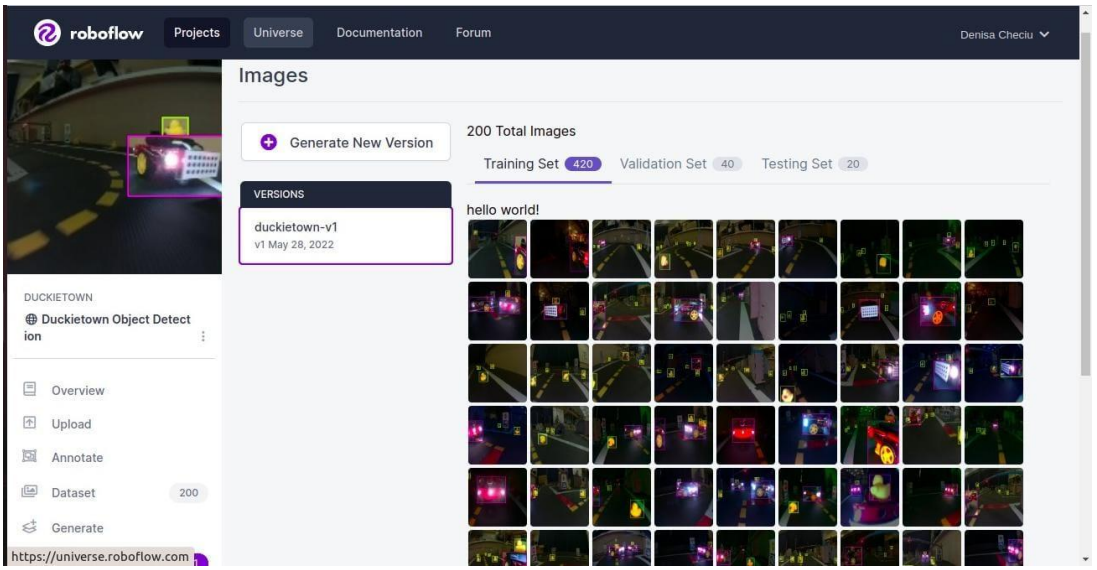

Figure 4.1.3 : Using the Roboflow software to generate our final data set

In the end, our training set was composed of 420 images, our validation set of 40 images and the training set of 20 images.

4.1.2 Object Identification Algorithm

Furthermore, having our data sets ready, we performed object detection using Yolov5, a novel convolutional neural network (CNN) that detects objects in real-time with great accuracy. This approach uses a single neural network to process the entire picture, then separates it into parts and predicts bounding boxes and probabilities for each component. These bounding boxes are weighted by the expected probability. The method "just looks once" at the image in the sense that it makes predictions after only one forward propagation run through the neural network. It then delivers detected items after non-max suppression (which ensures that the object detection algorithm only identifies each object once).

Below you can see a screenshot of the completed training that we performed on our dataset. For our 420 images, using 100 epochs, the process took 4.916 hours.

```
     95/99        0G   0.03738   0.02389  0.001815   0.06308        25       448: 100%|| 27/27 [01:44<00:00,  3.86s/it]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100%|| 2/2 [00:04<00:00,
                 all          40         154       0.914       0.801       0.916        0.44

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     96/99        0G    0.0375   0.02449   0.00225   0.06424        18       448: 100%|| 27/27 [01:36<00:00,  3.57s/it]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100%|| 2/2 [00:04<00:00,
                 all          40         154       0.872       0.838        0.91       0.438

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     97/99        0G   0.03691   0.02499  0.002036   0.06394        10       448: 100%|| 27/27 [01:47<00:00,  4.00s/it]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100%|| 2/2 [00:04<00:00,
                 all          40         154       0.916       0.826       0.896        0.43

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     98/99        0G   0.03729   0.02565  0.001705   0.06464        29       448: 100%|| 27/27 [02:07<00:00,  4.72s/it]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100%|| 2/2 [00:06<00:00,
                 all          40         154       0.872       0.848        0.89       0.432

     Epoch   gpu_mem       box       obj       cls     total   targets  img_size
     99/99        0G   0.03615   0.02544  0.002499   0.06409        31       448: 100%|| 27/27 [04:19<00:00,  9.62s/it]
               Class      Images     Targets           P           R      mAP@.5  mAP@.5:.95: 100%|| 2/2 [00:12<00:00,
                 all          40         154       0.912       0.822       0.895       0.448
              duckie          40         113       0.852       0.717       0.844       0.385
           duckiebot          40          41       0.973       0.927       0.945        0.51
Optimizer stripped from runs/train/yolov5s_results/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/yolov5s_results/weights/best.pt, 14.8MB
100 epochs completed in 4.916 hours.
```

Figure 4.1.4 : Terminal Screenshot of completed training

Figure 4.1.4 : Results of object recognition

4.2 Displacement Computation

For this part, we wanted to use the camera to compute the shortest distance from our duckiebot to the identified object. This was done using a separate python script which, after being run, printed the shortest distance to the console in cm.

The code was ideally designed for computing the distances from other duckiebots (specifically, using the back of a duckiebot). However, it must be noted that this computation only works when the duckiebot is the largest object in the frame (no integration to the detection part). It was implemented like this because since this feature was meant to be developed as a safety measure to avoid collisions, the distance only becomes relevant when the object is close.

The actual code was implemented using a Canny Edge detector to identify the contours, select the largest one and then compute the bounding box. The code can be reviewed in the included Github repository.

Some of the few further improvements would be:

1. Acquiring a camera with depth parameter
2. Use the localization sensors as in DB21

5. Further implementations

We believe our project offers a basis for other projects in the field of autonomous driving. Some examples could be the following:

Ability to detect the closest charging station and drive to it when needed.

Detecting when we are getting too close to a car/traffic light (any object that has importance in the system) using our displacement computation.

Driving to the closest client (duckie in our case) on a map who requests a ride.

Find the nearest pedestrian zone or parking spot to stop, or drop off the passenger.

6. Conclusions

After multiple sessions of testing the model built, it had a high accuracy rate on real-time object recognition of static or dynamic objects. This could clearly be seen on the attached video where after an object is identified, a label is generated and displayed identifying the object (either a duckiebot or duckie in our case).

For further information, refer to the video with documented results.

7. Project Contributions

Setting up and calibrating robot - Assylbek
Object recognition algorithm - Denisa
Displacement Computation - Rayan
Presentation - Denisa and Rayan
Report - Denisa, Rayan, Assylbek