



DUCKIETOWN

Object Identification and Displacement Computation

Team members: Assylbek, Denisa, Rayan
Group 2

RIS Project - Duckietown



DUCKIETOWN

Content

1. Introduction
2. Problem Formulation
3. Approach
 - a. Object identification
 - b. Displacement computation
4. Further implications
5. Results - Video



DUCKIETOWN

Introduction & Problem Formulation

Introduction



DuckieBots are low-cost mobile robots that are built almost entirely from off-the-shelf parts.

Team : Denisa, Rayan, Assylbek

Part of RIS Project class, based on the MIT 2016 project Duckietown



DuckieTowns are the urban environments: roads, constructed from exercise mats and tape, and the signage which the robots use to navigate around.



DUCKIETOWN

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.





DUCKIETOWN

Approach



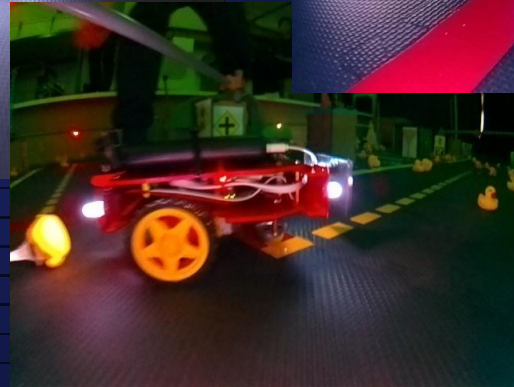
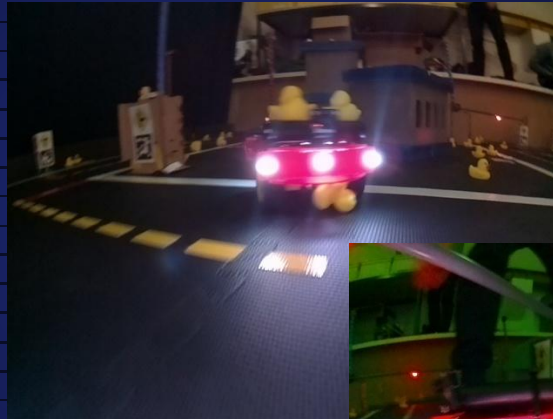
DUCKIETOWN

Object Detection

Training Data Preparation



1. Access a dataset containing images from logs of a Duckiebot in a Duckietown in a variety of lighting conditions with objects.

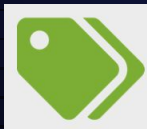
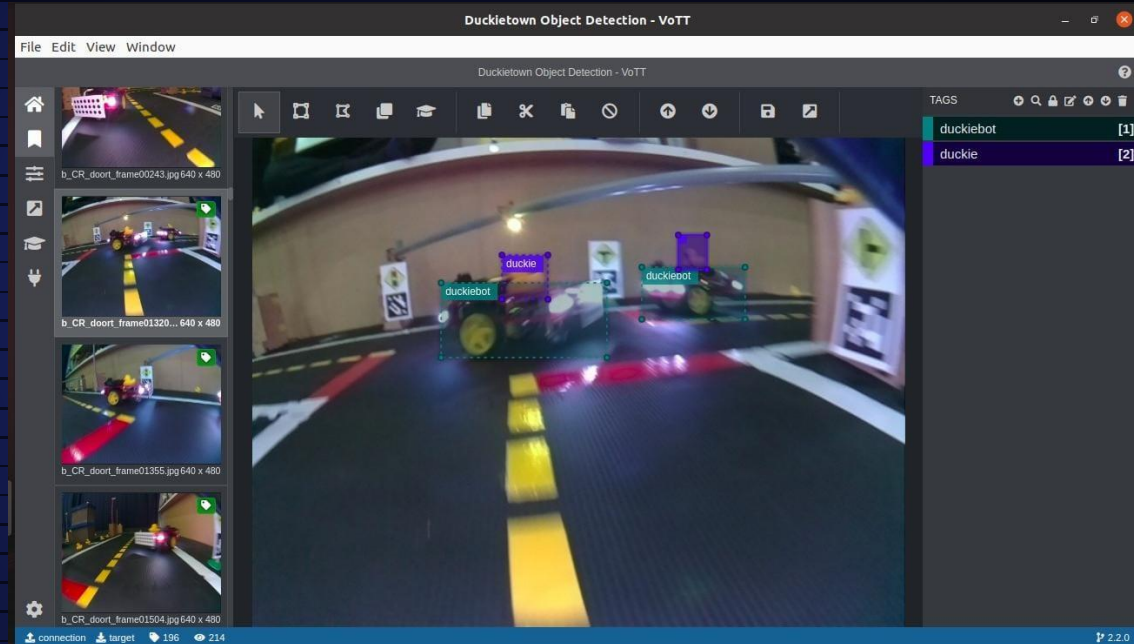


https://github.com/duckietown/duckietown-objdet/tree/master/duckie_data/training-images

Training Data Preparation



2. Annotate each image (using VoTT) by labelling each object type we later want to detect, using boxes to delimitate the margins.



VoTT



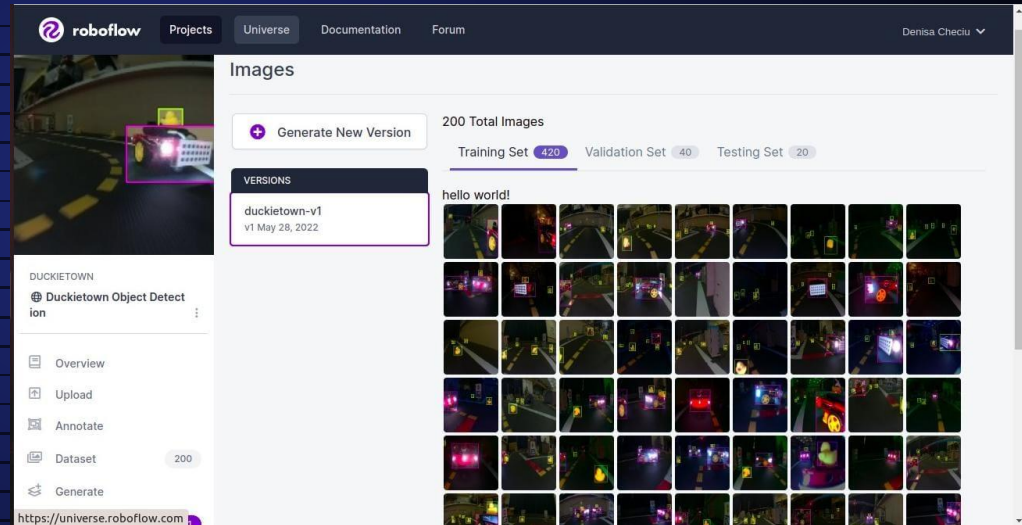
Training Data Preparation



3. Perform data collection with the annotated images (using Roboflow). In order to increase the accuracy of the algorithm, we generated a bigger dataset. By blurring out or zooming in on the already annotated images, we were able to create new training data.

Blurring - prepares for different camera focus/quality

Zoomin/out - prepares for different angles or proximity to detected objects



roboflow



Object Detection



For the object detection we used 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).


YOLOv5

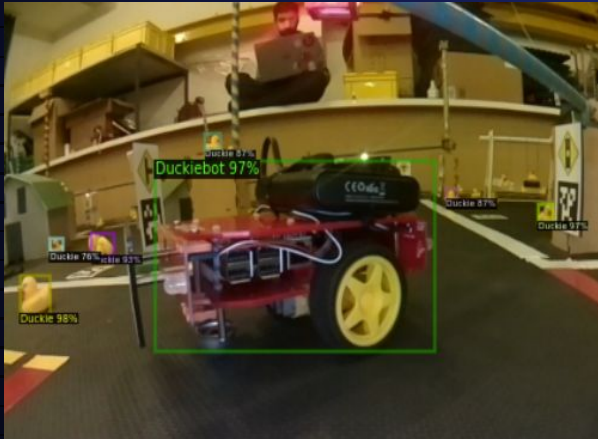
“You Only Look Once”



Object Detection Approach

Initially considered approaches:

1. HSV Thresholding
2. Convolutional Neural Network 



- Access a dataset containing images from logs of a Duckiebot in a Duckietown in a variety of lighting conditions with objects.
- Annotate each image (VoTT) by boxing each object type we later want to detect.
- Perform data collection with the annotated images (Roboflow).
- Use the YOLOv5 (novel convolutional neural network that detects objects in real-time with great accuracy) to train our algorithm and later perform object detection for duckies and duckiebots.



DUCKIETOWN

Displacement Computation



Displacement Computation

- Once we detect the object, we want to use the camera to compute the shortest distance from our duckiebot to the identified object.
- We did this by using a separate python script which, after being run, printed the shortest distance to the console.





DUCKIETOWN

Displacement Computation

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.



DUCKIETOWN

Displacement Computation

The actual code was implemented using a Canny Edge detector to identify the contours, select the largest one and then compute the bounding box.





DUCKIETOWN

Further implications of our project



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



Assistance in autonomous driving by detecting when we are getting too close to a car/object using our displacement computation.



Possibility of further implementing an autonomous cab system - driving to the closest passenger (duckie in our case) who needs a ride.



DUCKIETOWN

Results... (video)

Thank you!

