CONSTRUCTOR UNIVERSITY

Duckietown Project:

Path planning and object detection

By Group 5: Tayyab Butt

Spring 2023

Abstract:

Using the duckietown software, our object is to explore the ideas of path planning and object detection, primarily focusing on path planning. Our method for path planning uses RRT* algorithm, in our case we used a predefined map to simplify the process. For object detection we implement the commonly used YOLO framework where we use images of rubber duckies to fine-tune the model.

1. Introduction

Duckietown is a simulation of an urban environment, i.e. roads made from black mats and tape, obstacles are either mock-up trees or buildings, pedestrians are the yellow duckies and the duckiebots are the autonomous vehicles. As this is supposed the represent a real city, it is necessary to test how the vehicle reaches from a starting position to a destination. Due to the lack of a human driver, it is also important to be able to detect objects in front of the vehicle and identify them.

2. Objectives

This Project has 2 goals. First is to make a map of duckietown (or any alternative) and have the duckiebot plan the best path from point A to B. It is important to make this process as efficient as possible so that we are receiving the optimal path from start to finish. We use RRT* algorithm to find the best possible solution.

The second goal is to integrate an object detection into the duckiebot. This requires us to create an algorithm that can successfully detect rubber ducks and identify it with a box around it. For this we use OpenCV as it is a very popular image processing tool and use those images to train our model, for which we use the YOLOv3 framework. YOLO uses a convolution neural network (CNN) to process input images as an array of data to recognize repeating patterns in the images to identify the object it is meant to detect.

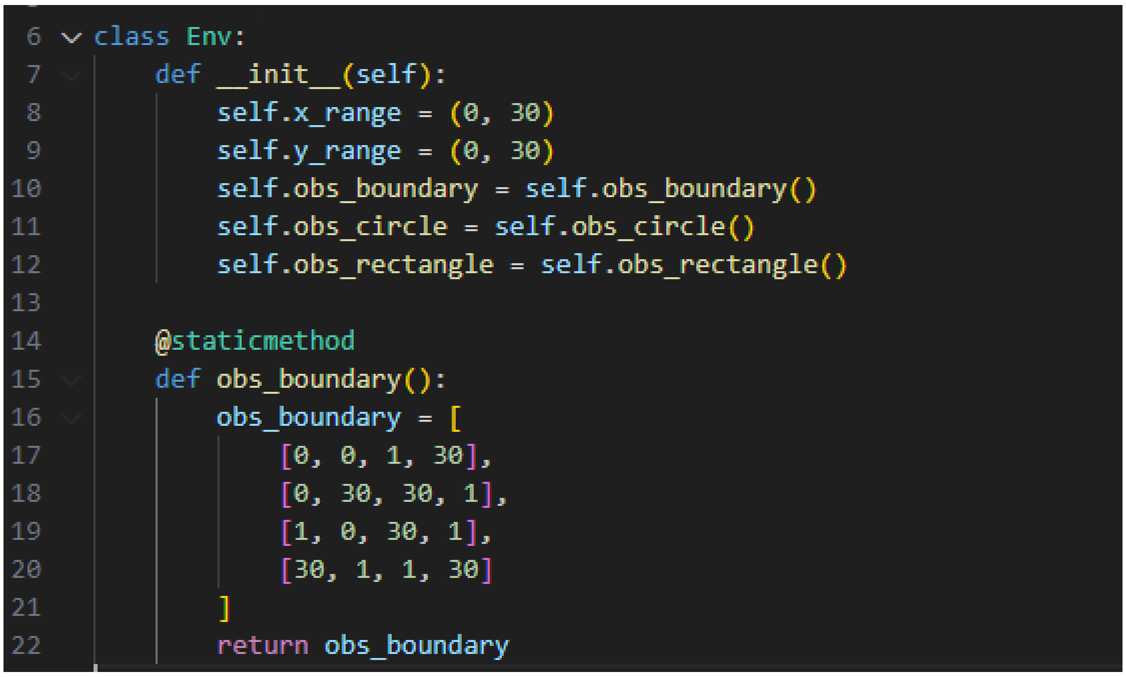
3. Approach

I. Path Planning

We start our path planning by creating a map of the environment. This simplifies the path planning process by already giving us the map that the robot is to use. We then implement the Rapidly-Exploring Random Trees (RRT*) Algorithm on the map we created for the duckiebot to find the shortest path from A to B.

The Tree starts with one node and starts creating many nodes moving away from the start position. It continues to branch out at random until it reaches the final position and then the algorithm is fine-tuned to output the shortest journey.

We have to define an environment for the duckiebot to move through and then implement the RRT* algorithm on the duckiebot using the predefined environment.



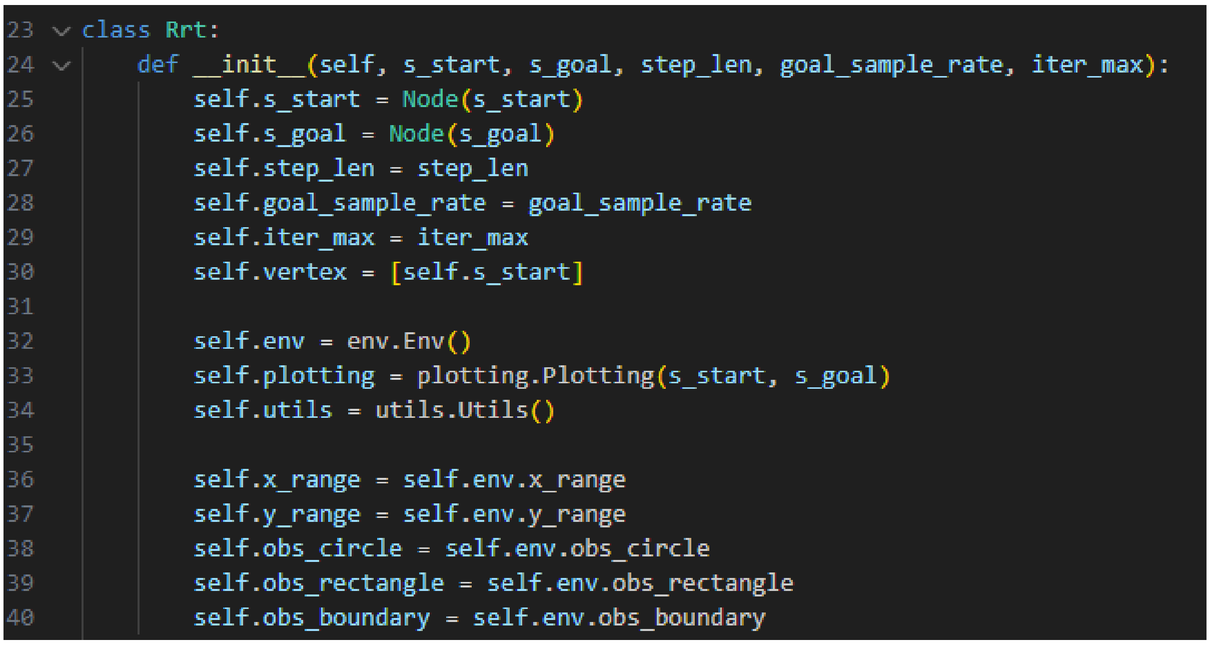
```python
class Env:
    def __init__(self):
        self.x_range = (0, 30)
        self.y_range = (0, 30)
        self.obs_boundary = self.obs_boundary()
        self.obs_circle = self.obs_circle()
        self.obs_rectangle = self.obs_rectangle()

    @staticmethod
    def obs_boundary():
        obs_boundary = [
            [0, 0, 1, 30],
            [0, 30, 30, 1],
            [1, 0, 30, 1],
            [30, 1, 1, 30]
        ]
        return obs_boundary
```

Fig 1.1: defining the environment and boundary.

After the environment is defined we can add obstacles and define them using the same method and placing their locations until we create a complete simulated environment with obstacles.

We also define a node and edge as that will be what expands the tree and then define RRT algorithm in the below snippet.



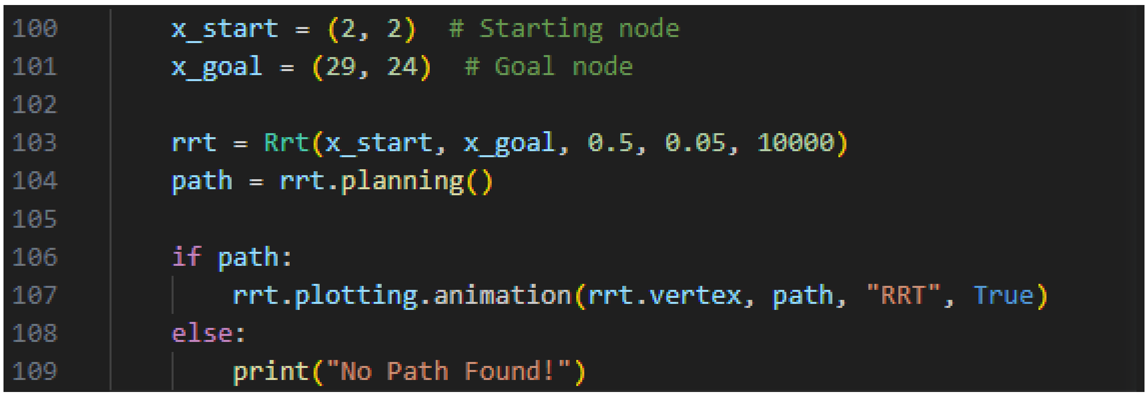
```python
class Rrt:
    def __init__(self, s_start, s_goal, step_len, goal_sample_rate, iter_max):
        self.s_start = Node(s_start)
        self.s_goal = Node(s_goal)
        self.step_len = step_len
        self.goal_sample_rate = goal_sample_rate
        self.iter_max = iter_max
        self.vertex = [self.s_start]

        self.env = env.Env()
        self.plotting = plotting.Plotting(s_start, s_goal)
        self.utils = utils.Utils()

        self.x_range = self.env.x_range
        self.y_range = self.env.y_range
        self.obs_circle = self.env.obs_circle
        self.obs_rectangle = self.env.obs_rectangle
        self.obs_boundary = self.env.obs_boundary
```

Fig 1.2: RRT algorithm definition

After that we can define a start and goal position and use the RRT function defines above to start the process.



```python
    x_start = (2, 2)  # Starting node
    x_goal = (29, 24)  # Goal node

    rrt = Rrt(x_start, x_goal, 0.5, 0.05, 10000)
    path = rrt.planning()

    if path:
        rrt.plotting.animation(rrt.vertex, path, "RRT", True)
    else:
        print("No Path Found!")
```

Fig 1.2: RRT algorithm used on start and goal positions.

II. Object Detection

For object detection, we start by gathering images of the objects we want to use as training data and then we label the images using LabelImg which is a popular tool used for image labelling. We label the image by localizing the object creating a bounding box around the object. After we have collected our dataset we then can preprocess the dataset using OpenCV to resize the image to an input size and reducing noise and split the dataset into a training and testing sets to evaluate the performance of the model.

Then, we can fine tune our model using a pretrained dataset such as the COCO dataset which is a famous object detection dataset. We implement transfer learning onto the COCO dataset by replacing the final classification layer with a layer that matches our desired classification (rubber duckie class). After we have trained our model we use it on the duckiebot to for inference.

4. Architecture

DTS (DuckieTown Shell): DTS is a python based utility for duckietown. It uses docker containers so that the user does not have to write a long docker run command line.

RRT* Algorithm: RRT* is a path planning algorithm that expands using nodes and edges in the shape of a tree to create and edge in all possible directions to maximize the chances of finding the goal position.

ROS: is a set of libraries and software tools that help in building robot applications.

OpenCV: is an open source library which offers a wide range of video/image processing functions. We use it to preprocess our images i.e. resize and blending.