RIS Project
Path planning and Object detection

By: Tayyab Butt
Group 5

Our Bot

DB21M

jaybot

Contents

Path Planning

Path planning is a crucial aspect of autonomous robotics. It involves finding an optimal path from a starting point to a destination while avoiding obstacles. Some common algorithms for finding the most optimal path is A*, Dijkstra and RRT* algorithms.

In this presentation I will be using RRT* (Rapidly-exploring Random Trees) to find the optimal path for ‘’ jaybot’’ to reach his destination.

Object Detection

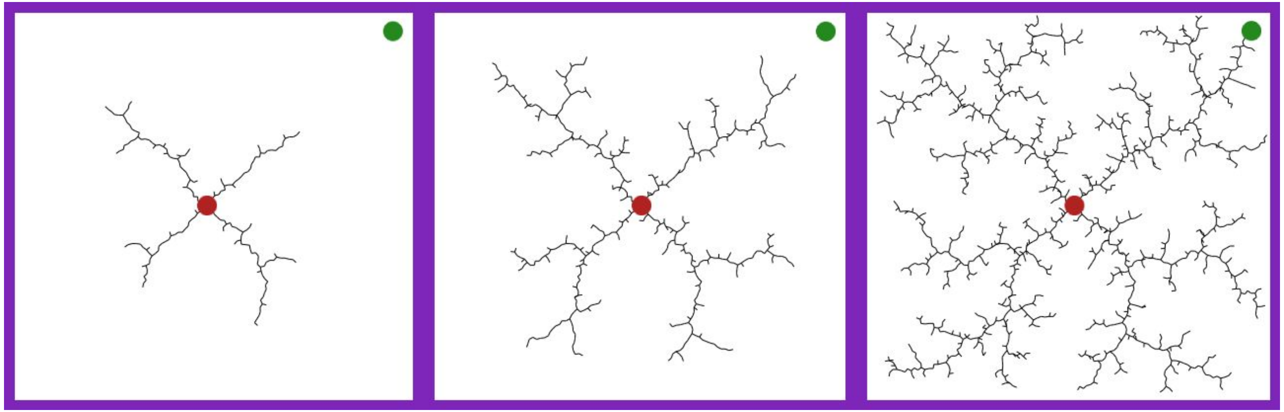
Object detection is a fundamental task in computer vision. The primary objective is to accurately recognize and locate specific objects of interest within a given image or video.   I am using the YOLO framework which is known for its speed and accuracy.

The aim is to use object detection to enable jaybot to detect rubber duckies in the camera’s field of view and correctly identify them.

Implementation of RRT* Algorithm

RRT (Rapidly-exploring Random Tree) algorithm is a popular motion planning algorithm used to find feasible paths for robots in complex environments by constructing a tree-like data structure - consisting of nodes and edges - to represent the search space.

To explain in detail; we start by defining the environment as a class and the boundary of our map, we also define some obstacles (since we are mapping in 2-D we can define obstacles as basic shapes). We then define how the plotting will be done by introducing nodes and edges and defining their characteristics. In the end we define the Rrt function and test it on a simulated environment.



Code snippets

```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
```

Environment class with obstacles

```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
```

RRT* algorithm class

Code snippets

```python
    def planning(self):
        for i in range(self.iter_max):
            node_rand = self.generate_random_node(self.goal_sample_rate)
            node_near = self.nearest_neighbor(self.vertex, node_rand)
            node_new = self.new_state(node_near, node_rand)

            if node_new and not self.utils.is_collision(node_near, node_new):
                self.vertex.append(node_new)
                dist, _ = self.get_distance_and_angle(node_new, self.s_goal)

                if dist <= self.step_len and not self.utils.is_collision(node_new, self.s_goal):
                    self.new_state(node_new, self.s_goal)
                    return self.extract_path(node_new)

        return None
```

Path planning function

```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!")
```

main() function with start and goal positions defined

Steps for Object Detection

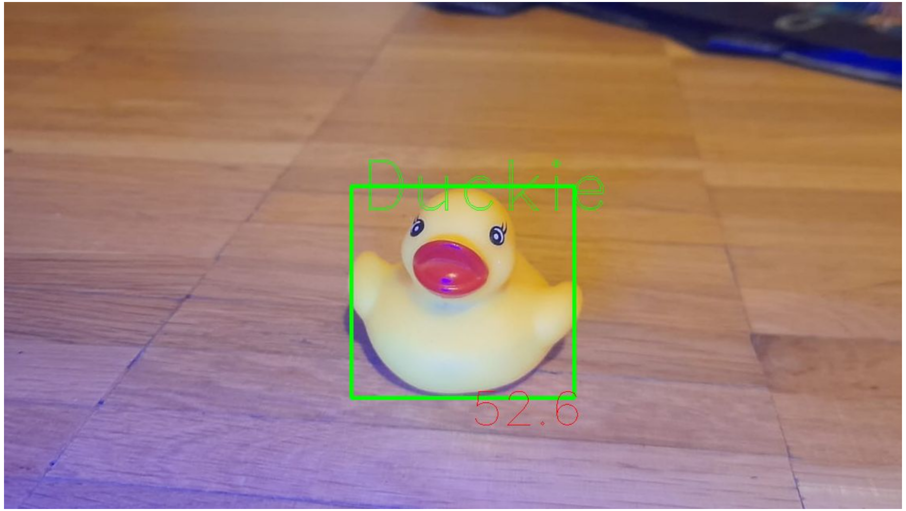
- Collect images dataset (images of duckies were taken).
- Preprocessing of images where images are labelled and resized.
- Object detection algorithm using pretrained dataset
- fine -tuning of threshold to obtain best results
- Output showing box around desired object with probability of prediction

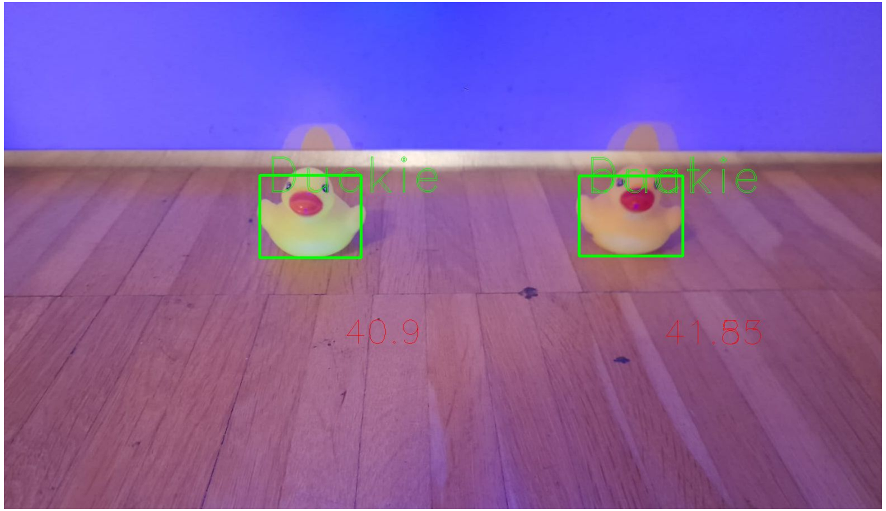
Code snippets

```python
# Path to the frozen inference graph and label map file
PATH_TO_FROZEN_GRAPH = 'C:\Users\Tayyab\OneDrive\Desktop\Path planning\frozen_inference_graph.pb'
PATH_TO_LABELS = 'C:\Users\Tayyab\OneDrive\Desktop\Path planning\label_map.pbtxt'

# Load the frozen TensorFlow model
detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.compat.v2.io.gfile.GFile(PATH_TO_FROZEN_GRAPH, 'rb') as f:
        serialized_graph = f.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.compat.v1.import_graph_def(od_graph_def, name='')

# Load the label map
label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map, max_num_classes=90, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
```

```python
            # Visualize the detection results
            vis_util.visualize_boxes_and_labels_on_image_array(
                frame,
                np.squeeze(boxes),
                np.squeeze(classes).astype(np.int32),
                np.squeeze(scores),
                category_index,
                use_normalized_coordinates=True,
                line_thickness=4,
                min_score_thresh=0.5)

            # Display the resulting frame
            cv2.imshow('Object Detection', frame)

            # Exit if 'q' is pressed
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
```

Object detection results





Object detection from phone camera (reason explained below)

Software Dependencies

1. Path Planning
    a. Python: Programming languages used for implementing algorithms
    b. ROS framework for building for building application
    c. Numpy and Matplotlib: libraries for numerical computation and data visualization in python
2. Object Detection
    a. Python
    b. YOLOv3: deep learning architecture using Darknet Backbone
    c. LabelImg: annotation tool for labelling and annotating images
    d. Tensorflow Object detection API
    e. OpenCV: computer vision library for image processing.

Results

As shown above, I was successfully able to use object detection to detect the duckies however, I was not able to access any footage from the camera of the duckiebot. Hence, I could only take images for object detection with my phone camera. During the course of this project and also until the end I faced many issues with the duckiebot and until now it fails to start up.

I replaced the bot and despite my relentless effort I could not successfully complete first boot. I have attached a video below showcasing what I experience when the jaybot is powered on.

This video was taken on 2 day before submission of this presentation. It shows the duckiebot 20 mins after the power button had been turned on. The display is not on and the computer does not detect it. (This is the second bot provided to me after the previous one output the same issue).

Conclusion/What was learned

In conclusion, this project was a great learning opportunity for me as I now have a very practical experience with ROS and machine learning. I also gained an intuitive understanding of how computers talk to robots and got a look into the hidden layers behind the curtains.

Unfortunately, I faced a lot of problems with our duckiebot near the end and failed to make it work however, given the right circumstances I wholeheartedly agree that I would be more than able to finish the project in the way that it was intended.