ELEC 6672: Autonomous Mobile Robots

Project # Final

Following the Lane

Pushpa Latha Vudatha

Alberto Avila

December 14th, 2021

Table of Contents

I. Introduction

Autonomous control of different robotic system, specifically cars, rely heavily on image processing to be able to recognize the road and obstacles and therefore control and steer the vehicle.

Goal:

The goal for this project is to be able to collect the video from the Duckiebot camera, analyze the images, detect the left and right boundary of the lane and control itself to be able to follow the detected lane.

Expected Outcomes

To apply the previous learned image processing and edge detection techniques dynamically through the processing of video. This project will explore further capabilities of the Duckiebot such as the collection and processing of video as it is moving. Furthermore, this project will require the use of control algorithms to be able to adjust the trajectory of the robot.

Task Description

Write the code so the Duckiebot can follow the lane on the map set up on the classroom (see image 1). The grading will be assessed 7by the distance the robot will be able to achieve (see image 2).

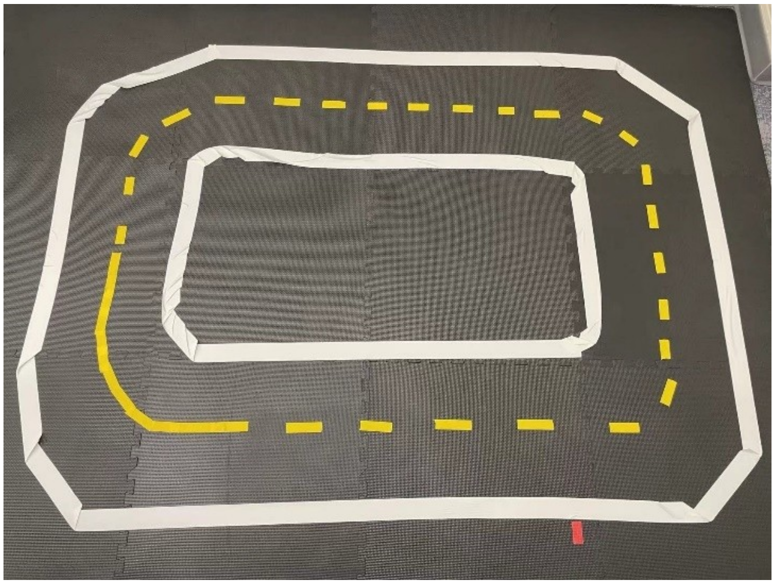

Figure 1 - Follow the Lane Map

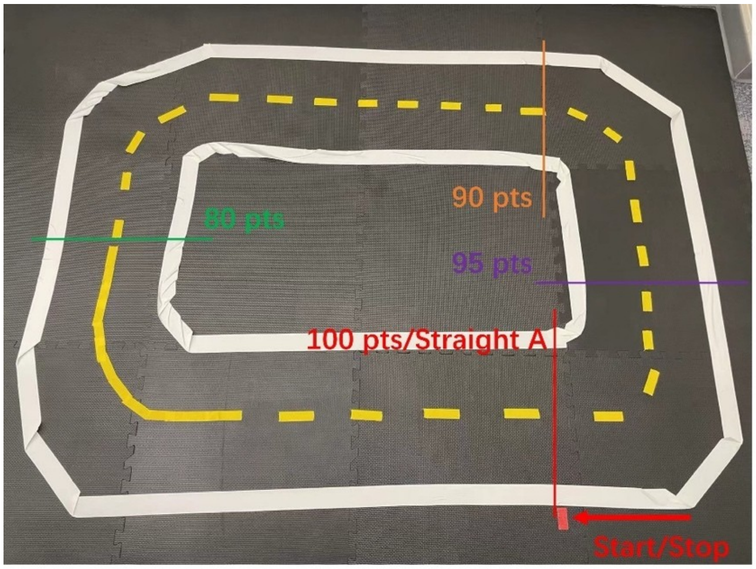

Figure 2 - Project Rubric

II. Calibration

Upon the start of the trials for this project and us having our robot move, it was apparent that the expected movement was not at all “expected” as the robot kept steering the wrong direction. At this point, we found that calibration was necessary in order for the Duckiebot to move as expected. Following operational manual Unit C-12.

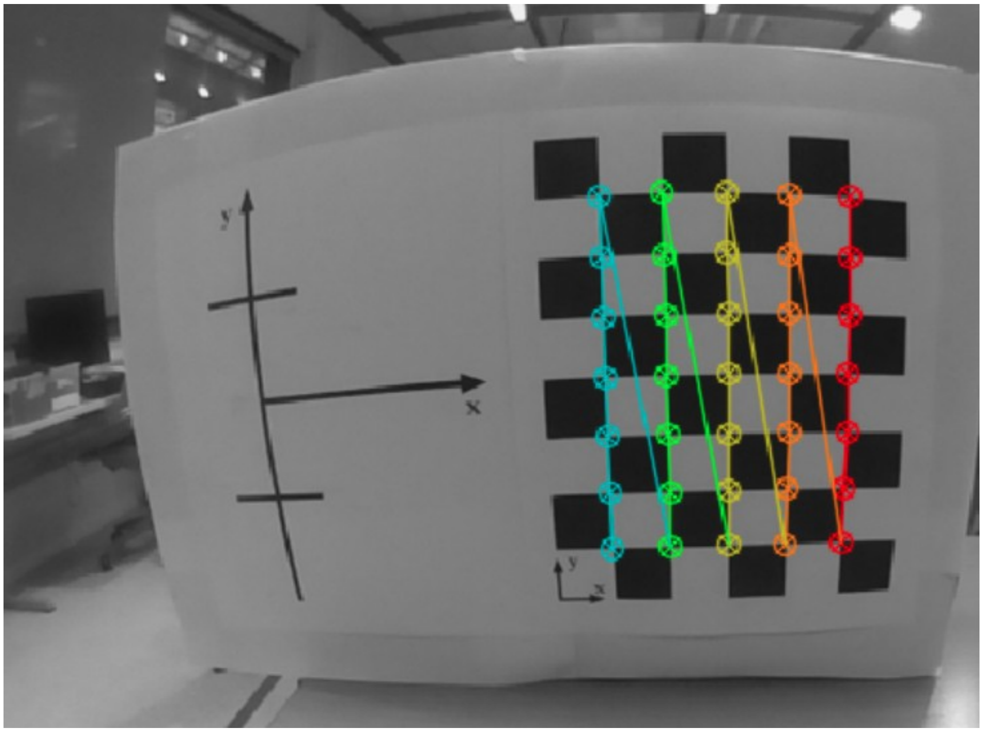

Figure 3 - Camera Calibration Screen

Our methodology was to leave the robot still and move the calibration chart until all the parameters were fully green.

III. Video Capturing

Using the utilities from ROS on the Duckiebot, by accessing the topic of /camera_node/image/compressed in order to capture the images from the robot. This utility allows the video to be accessed in the form of images. This images are constantly fed into this container.

The images then get adjusted for size and cropped for useful dimensions. The size is adjusted to prevent delay on the image analysis due to the fact of the resolution of the robot’s camera. Additionally, the image get’s cropped as the interest area on the image/video, the lane, is the only one important to detect our direction.

IV. Line Detection

As per our previous assignment, we were able to primarily use the Canny edge detection to be able localize the edges of the line.

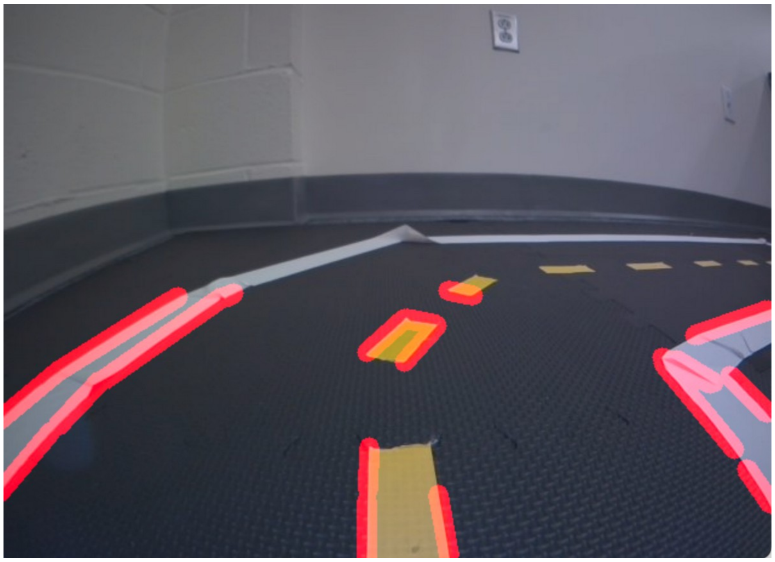

Figure 4 - Turn Line Detection

Figure 5 - Straight Line Detection

For our line detection we also calculate a single slope by taking the average of all the line edge that were found during the line detection. We do a similar where we take the average of the angle of all the lines. Each line is calculated based on the angle of a first order polynomial formed by the lines detected.

```
def angle_of_line(x1, y1, x2, y2):
    parameters = np.polyfit((x1, x2), (y1, y2), 1)
    slope = parameters[0]
    y_intercept = parameters[1]

    myradians = math.degrees(math.atan2(-y1-y2, x2-x1))
    #mydegrees = math.degrees(myradians)
    #myradians = math.radians(mydegrees)
    return myradians, slope
```

Code 1

```python
def average_lines(frame, lines):
    # Empty arrays to store the coordinates of the left and right lines
    left = []
    right = []
    # Loops through every detected line
    if lines is None:
        return np.array([[0,0,0,0], [0,0,0,0]])

    for line in lines:
        # Reshapes line from 2D array to 1D array
        x1, y1, x2, y2 = line.reshape(4)
        # Fits a linear polynomial to the x and y coordinates and returns a vector of coefficients which describe the slope and y-intercept
        parameters = np.polyfit((x1, x2), (y1, y2), 1)
        slope = parameters[0]
        y_intercept = parameters[1]
        # If slope is negative, the line is to the left of the lane, and otherwise, the line is to the right of the lane
        if slope < 0:
            left.append((slope, y_intercept))
        else:
            right.append((slope, y_intercept))
    # Averages out all the values for left and right into a single slope and y-intercept value for each line
    left_avg = np.average(left, axis = 0)
    right_avg = np.average(right, axis = 0)
    # Calculates the x1, y1, x2, y2 coordinates for the left and right lines
    left_line = get_line_coordinates(frame, left_avg)
    right_line = get_line_coordinates(frame, right_avg)
    return np.array([left_line, right_line])
```

Code 2

V. Trajectory & Control

We are doing the trajectory control based on the slope of the lanes that we get from lane detection method. Lane detection method calculates the average of all the line segments on for both lists the left side and the right side. After obtaining the two lines we take the average of the two to get a center line coordinates, After which we get the average slope and y-intercept of the center line, which the robot uses to move.

We captred static images of the lane at certain locations and calculated lane detection on these images to observe the results of lane detection. The static images are refered in the Table 1. We observed slope of the line for for robot movements and took logs of the movemnets to understand how the robot is behaving for a certain slope and linear velocity. From the logs, refered in Table 1, we observed that the robot has been moving with a slope > 0 to go straight and slope < 0 for taking right turn. This slopes varry from robot to robot as the calibtration differs from each robot.

S.No	Image	Slope
1		-0.242
2		2.281
3		-1.801
4		-0.635

5		0.036

Table 1

We implemented the control logic based on the slope from lane detection. With a line slope > 0 we made the robot to go straight and if slope < 0 we made the robot to take right turn. Now we know which direction the robot should take. But how much linear velocity and angular velocity should the robot receive?? To understand that we ran '***dts duckiebot keyboard_control***' from host computer and checked logs on

rostopic echo /<Duckiebot Name>/joy_mapper_node/car_cmd

Table 2 shows the value of linear velocity (v) and angular velocity (omega) for the keyboard actions. From the Table 2 we understand that in-order to make the robot move in straight in forward direction we should set the v value ~0.409 and angular velocity 0.0 and to make the robot take right turn we should set the v value 0.0 and angular velocity -8.3. After trying out these value with v = 0.4 the robot moved too fast. Hence we set v to 0.06, which worked after numerous velocity trials. Omega ~ -8.3 made the robot to take a deep right turn but in our case the robot should take the right turn gradually. And the omega to 0.0 did not made the robot to move straight as there will error from calibration and will be error accumulated in each move. Omega value of 0.4, v value of 0.06 worked for forward move in a straight line. We figured out omega as -0.9 to take right turn, which worked after numerous omega trials. Robot moving direction changes with a slight change in the wheel postion. To allleviate this problem we made the angular velocity to 0.41 and 0.32 alternatively so that the movement is balanced. As the FPS is more, the robot moves fast, hence we publised control message for every 5 frames.

S.No	Keyboard control	Linear velocity (v)	Angular Velocity (Omega)
1	Forward	0.4099999964237213	0.0
2	Backward	-0.4099999964237213	0.0
3	Left	0.0	8.300000190734863

4	Right	0.0	-8.300000190734863

Table 2

```
########### write the control code ################
########### output:  u=[w, v]
self.counter +=1
#sample function, delete it
self.phi_error = angle
v = 0.06

# forward
if slope >= 0:
    if self.pub_c % 2 ==0:
        om = 0.41
    else:
        om = 0.32
# right turn
else:
  om = -0.9

u=[v, om]
#print ('------ u ', u , slope, angle)
if self.counter %5 ==0:
   print ('------------Publishing.......')
   self.pub_c += 1
   self.publish_command(u)
```

Code 3

```
def publish_command(self, u):
    """Publishes a car command message.
    Args:
        u (:obj:`tuple(double, double)`): tuple containing [v, w] for the control action.
    """
    print ('Publishing...........', u)
    car_control_msg = Twist2DStamped()
    car_control_msg.header.stamp = rospy.Time.now()

    car_control_msg.v = u[0]  # v

    car_control_msg.omega = u[1]  # omega

    self.pub_car_cmd.publish(car_control_msg)
```

Code 4

VI. Conclusion

This project was a great combination of everything that was taught throughout the semester. This project allowed the further integration of ROS nodes and topics in conjunction of image processing to aid in the control of the robot.

This project brough with it many real-life challenges, as the hardware and control was not behaving as expected, and therefore forced us to look deeper into each of the functions that were used. Furthermore, this project took us to take a methodical look at code development, where each of the processes were tested at the time, and the next one was implemented and troubleshoot after each one is completed.

This project also allowed us to see the infinite possibilities that machine learning could bring into autonomous control, as the “normal” control approach showed how its inaccuracies. We had to manipulate, in some cases randomly, to be able to get a reasonable outcome.

VII. Code

Move.py

```python
#!/usr/bin/env python3

from lane_detection import *

import os
import cv2
import yaml
import time
import rospy
import numpy as np

from duckietown_msgs.msg import Twist2DStamped
from sensor_msgs.msg import CompressedImage
from std_msgs.msg import String

#import visual_servoing_activity
from duckietown.dtros import DTROS, NodeType, TopicType
from duckietown.utils.image.ros import compressed_imgmsg_to_rgb, rgb_to_compressed_imgmsg


class LaneServingNode(DTROS):
    """
    Performs a form of visual servoing based on estimates of the image-space lane orientation
    Args:
        node_name (:obj:`str`): a unique, descriptive name for the ROS node
    Configuration:
    Publisher:
        ~wheels_cmd (:obj:`WheelsCmdStamped`): The corresponding resulting wheel commands
    Subscribers:
        ~/image/compressed (:obj:`CompressedImage`):
            compressed image
    """

    def __init__(self, node_name):⋯


    def cb_action(self, msg):⋯

    def detect_lane_markings(self, image):⋯

    def p_feedback(self): #proportional-error feedback system⋯


    def cb_image(self, image_msg):⋯

    def publish_command(self, u):⋯

    @staticmethod
    def trim(value, low, high):⋯

    def angle_clamp(self, theta):⋯

    def read_params_from_calibration_file(self):⋯

    def on_shutdown(self):⋯


def rescale(a: float, L: float, U: float):⋯


if __name__ == "__main__":
    # Initialize the node
    encoder_pose_node = LaneServingNode(node_name="visual_lane_servoing_node")
    # Keep it spinning
    rospy.spin()
```

Lane_detection.py

```python
#!/usr/bin/env python3

import cv2 as cv
import numpy as np
import math

def edge_detection(frame):

# This function crops the Region Of Interest from the frame as we want to ignore the unnecessary image pixels
def mask_ROI(frame):

def average_lines(frame, lines):

def get_line_coordinates(frame, parameters):

def replace_yellow_lines (image):


def angle_of_line(x1, y1, x2, y2):

def draw_lines_left_right (frame, lines):


def draw_lines(frame, lines):

def draw_lines_p (img, linesP):

def detect_lanes (frame):

```

VIII. References

- ELEC6672:Autonomous Mobile Robots Documentation
- DUCKIETOWN.org - Documentation