



ELEC 6672: Autonomous Mobile Robots

Project # Final

Following the Lane

Pushpa Latha Vudatha

Alberto Avila

December 14th, 2021

Table of Contents

I. Introduction.....	3
Goal:.....	3
Expected Outcomes.....	3
Task Description.....	3
II. Calibration.....	5
III. Video Capturing.....	5
IV. Line Detection.....	6
V. Trajectory & Control.....	7
VI. Conclusion.....	10
VII. Code.....	12
Move.py.....	12
Lane_detection.py.....	13
VIII. References.....	13

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 by the distance the robot will be able to achieve (see image 2).

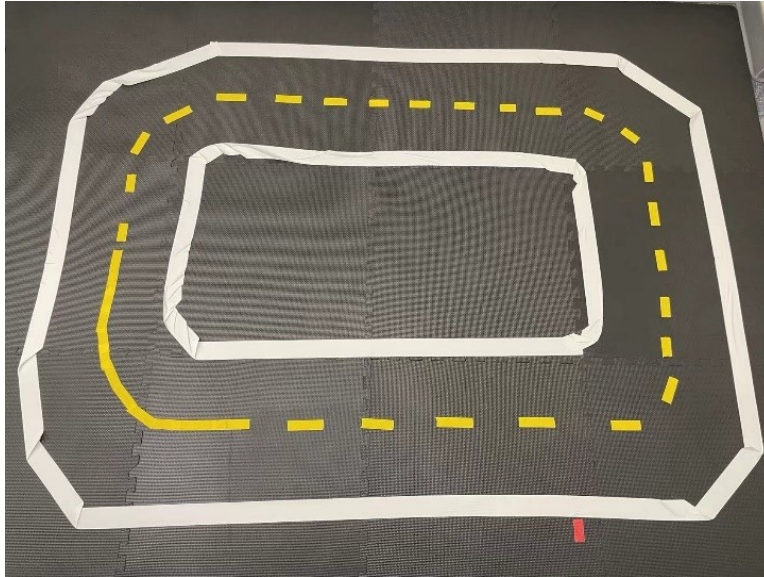


Figure 1 - Follow the Lane Map

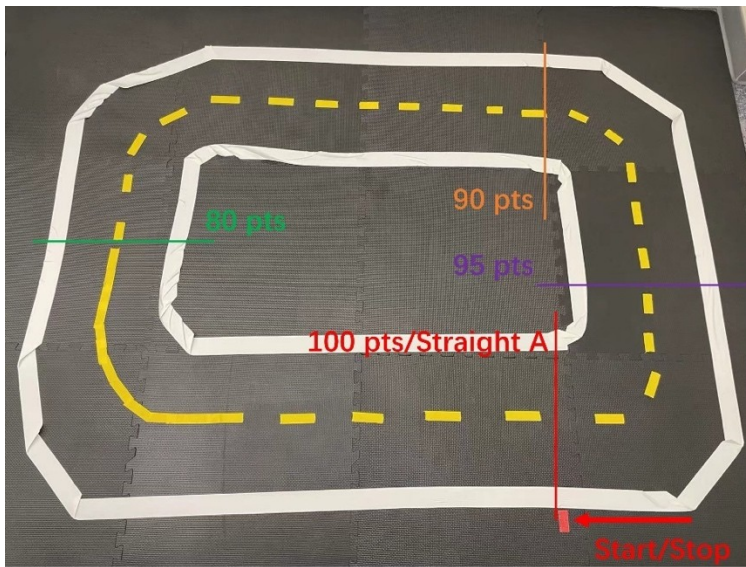


Figure 2 - Project Rubric

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.

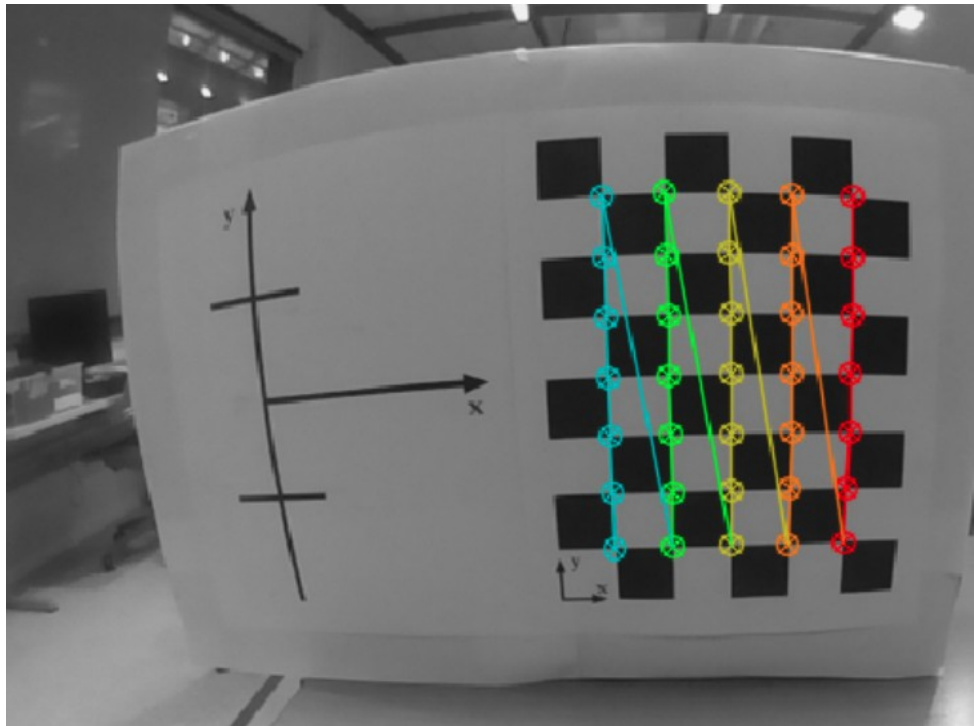


Figure 3 - Camera Calibration Screen

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. These 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 gets 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 to 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

```

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])

```




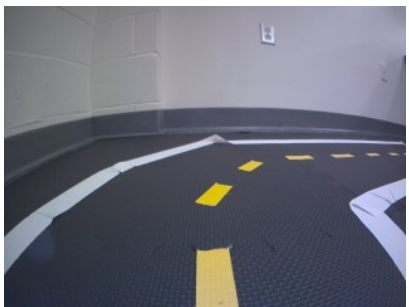
Code 2

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 captured 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 referred in the Table 1.

We observed slope of the line for robot movements and took logs of the movements to understand how the robot is behaving for a certain slope and linear velocity. From the logs, referred 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. These slopes vary from robot to robot as the calibration 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 '***dto 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 (ω) 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. $\omega \sim -8.3$ made the robot to take a deep right turn but in our case the robot should take the right turn gradually. And the ω 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. ω value of 0.4 , v value of 0.06 worked for forward move in a straight line. We figured out ω as -0.9 to take right turn, which worked after numerous ω trials. Robot moving direction changes with a slight change in the wheel position. To alleviate 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 published control message for every 5 frames.

S.No	Keyboard control	Linear velocity (v)	Angular Velocity (ω)
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 brought 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

```
1  #!/usr/bin/env python3
2
3  from lane_detection import *
4
5  import os
6  import cv2
7  import yaml
8  import time
9  import rospy
10 import numpy as np
11
12 from duckietown_msgs.msg import Twist2DStamped
13 from sensor_msgs.msg import CompressedImage
14 from std_msgs.msg import String
15
16 #import visual_servoing_activity
17 from duckietown.dtros import DTROS, NodeType, TopicType
18 from duckietown.utils.image.ros import compressed_imgmsg_to_rgb, rgb_to_compressed_imgmsg
19
20
21 class LaneServingNode(DTROS):
22     """
23     Performs a form of visual servoing based on estimates of the image-space lane orientation
24     Args:
25         node_name (:obj:`str`): a unique, descriptive name for the ROS node
26     Configuration:
27     Publisher:
28         ~wheels_cmd (:obj:`WheelsCmdStamped`): The corresponding resulting wheel commands
29     Subscribers:
30         ~/image/compressed (:obj:`CompressedImage`):
31             compressed image
32     """
33
34     def __init__(self, node_name):
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```

Lane_detection.py

```
1 #!/usr/bin/env python3
2
3 import cv2 as cv
4 import numpy as np
5 import math
6
7 def edge_detection(frame):=
17
18 # This function crops the Region Of Interest from the frame as we want to ignore the unnecessary image pixels
19 def mask_ROI(frame):=
34
35 def average_lines(frame, lines):=
62
63 def get_line_coordinates(frame, parameters):=
82
83 def replace_yellow_lines (image):=
97
98
99 def angle_of_line(x1, y1, x2, y2):=
108
109 def draw_lines_left_right (frame, lines):=
129
130
131 def draw_lines(frame, lines):=
140
141 def draw_lines_p (img, linesP):=
147
148 def detect_lanes (frame):=
181
182
```

VIII. References

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