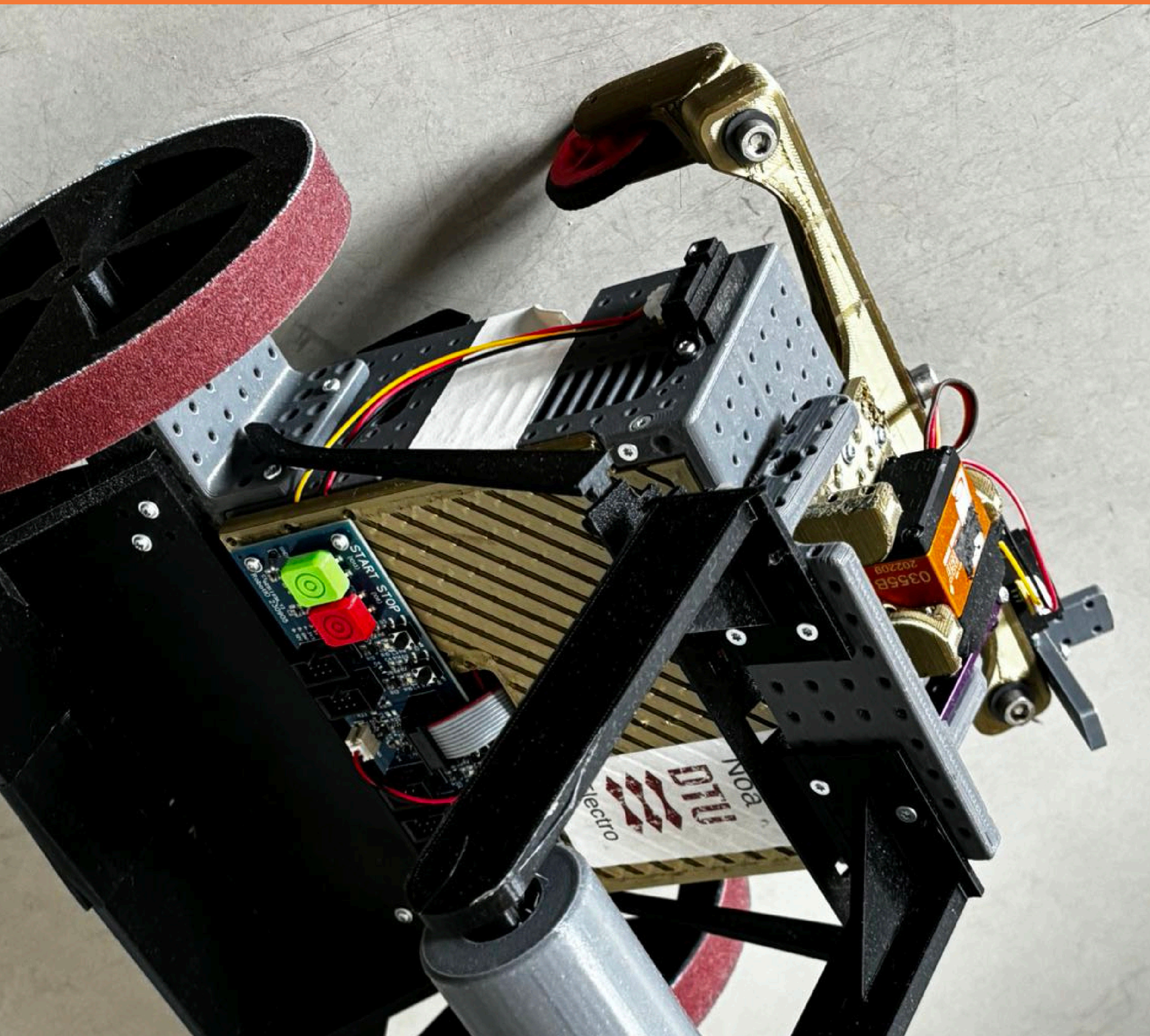
DTU ELECTRO
Department of Electrical and Photonics Engineering

Challenge report
Group 110 Noa

34755 Building dependable robot systems
Huan Hu
2024



Challenge report
Group 110 Noa

By
(s232179) Huan Hu

Course	34755 Building dependable robot systems
Level	Master (in Electrical Engineering)
Period	spring 2024
ECTS	5

Contents

1 Introduction

1.1 The group

The group initially consisted of seven members, and in the end, no one dropped the course, maintaining a steady count of seven throughout.

1. s232179 Huan Hu (me).
2. s232272 Zekun Wang (PMO).
3. s232172 Yuxuan Ma (CPO).
4. s232258 Xingyi Liu (CTO).
5. s232293 Dong Yun (SLO).
6. s232925 Yuansheng Zhou (HLO).
7. s232896 Yiming Zhang (CCO).

1.2 DTU Robocup performance

During the qualification round, we scored an impressive 14 points. However, in the finals, we encountered a minor hardware issue that prevented us from passing three gates, which resulted in us only obtaining 5 points.In fact, during our testing phase, we had already been capable of completing all tasks except for the vision part and anticipated securing 19 points. But that's the nature of competition, everything is uncertain. Reaching the finals was an achievement in itself, and I am quite content with that.

In fact, during our previous tests, we managed to get the car to complete the racetrack in under 10 seconds. However, due to a combination of factors such as the overly slick surface, which necessitated adding weight, as well as the addition of front wheel components and the constraints of our tuning time, we were unable to find a stable set of PID parameters that would allow our car to consistently complete the racetrack in under 10 seconds without veering off course after the significant modifications and increased weight. Therefore, we opted for a cautious approach during the race and chose a low speed line-tracking strategy.

1.3 Planning tools

We primarily relied on the Gantt chart and task list established at the beginning to monitor progress. Our weekly team meetings were also conducted very effectively, allowing everyone to discuss the details and approaches of task execution — a method far superior to inefficient individual thinking.

Additionally, the double diamond process proved to be quite effective. Brainstorming before execution yielded good results. Overall, these planning tools worked exceptionally well.

1.4 Team performance

Our team was performing well overall before the Easter break. However, anyone familiar with this course knows that the period around Easter is when the workload dramatically increases, accounting for nearly 90% of the total. During the Easter period, I, along with Zekun Wang, Yiming Zhang, and Dong Yun, handled all the coding, testing, debugging, fixture design and printing, and robot modifications. The others (Yuansheng Zhou, Yuxuan Ma, Xingyi Liu) were scarcely present, leaving the heavy workload on us.

In particular, Yiming Zhang spent all day in the library, working about 10 hours a day for roughly a week straight. I think it was incredibly unfair to him, as he ended up writing most of the code that the three absent team members were supposed to complete.

1.5 The robot

We started with robot 110 Noa. One day, the two motors all completely broke down, but fortunately, Christian fixed them. All two matches and all testings before and after were conducted on Noa; it is Noa that is on the cover page and on figure 1.1.

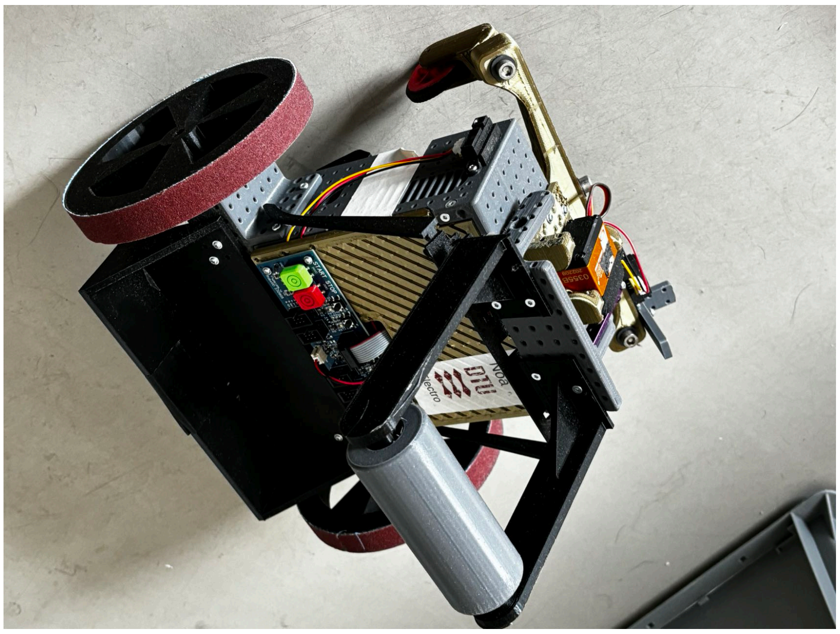


Figure 1.1: Robot 110 Noa (The ultimate version)

1.6 My challenge

My role was QAO (Quality Assurance Officer) and I participated in developing most challenges and provided suggestions for improvement.

The original allocated challenges are racetrack and closed tunnel. However, after Easter, since the four of us had to complete tasks normally assigned to seven people, I ended up participating in coding for almost every task.

In this report, I will discuss the task I worked on the most: the racetrack. To prevent someone from having no challenges to write about, I will only mention one challenge.

All data and results are obtained from ASTA and during the competition.

2

The challenge

2.1 Introduction

The most significant challenge of this task is to find a set of PID parameters that allow the our robot to exceed speeds of 1m/s while following a line, as well as to use a completely different set of PID parameters at turns, enabling the robot to navigate sharp corners and thus score all possible points. Additionally, posture correction is crucial when coming out of the closed tunnel and preparing to sprint.

2.2 Challenge analysis

The challenge will start after the robot has completed opening the doors and has exited the closed tunnel, with both doors of the tunnel remaining open. And the challenge is finished when the car passes the end gate near the finish line and then it will be ready for next challenge of closing all the door of the closed tunnel.

There are maximum 3 points on the racetrack if the time to complete the racetrack is within 10 seconds. Other scores are awarded as follows: 2 points for completing within 15 seconds, 1 point for completing within 20 seconds, and 0 points for times exceeding 20 seconds. All 3 points should be gained.

The first issue arises after our robot has left the closed tunnel and approaches the door, then rotates 360 degrees on the spot, preparing to enter the racetrack and pass through the start gate. At this point, the position of the car may be off-center, leading to an initial misalignment that cannot be corrected once acceleration begins.

The second consideration is that the racetrack, approximately 10 meters long, needs to be completed within 10 seconds. Choosing a speed of 1.2 meters per second for the straight sections is reasonable.

The third issue is that we need to make our robot very sensitive at the turns, capable of navigating large angles, to ensure that it does not veer off the track due to the high speeds on the straight sections.

Finally, given that the racetrack is composed of two straight sections and two sharp 90-degree corners, determining the criteria for switching between states (from straight to turn or from turn to straight) requires consideration. Since the shape and distance of the track are fixed, using distance sensors to measure the distance traveled and angle sensors to record the angles turned for state switching is a very reliable method.

2.3 Route & scheme planning

The challenge starts at A in figure 2.1, and the direction is oriented towards the start gate of racetrack.

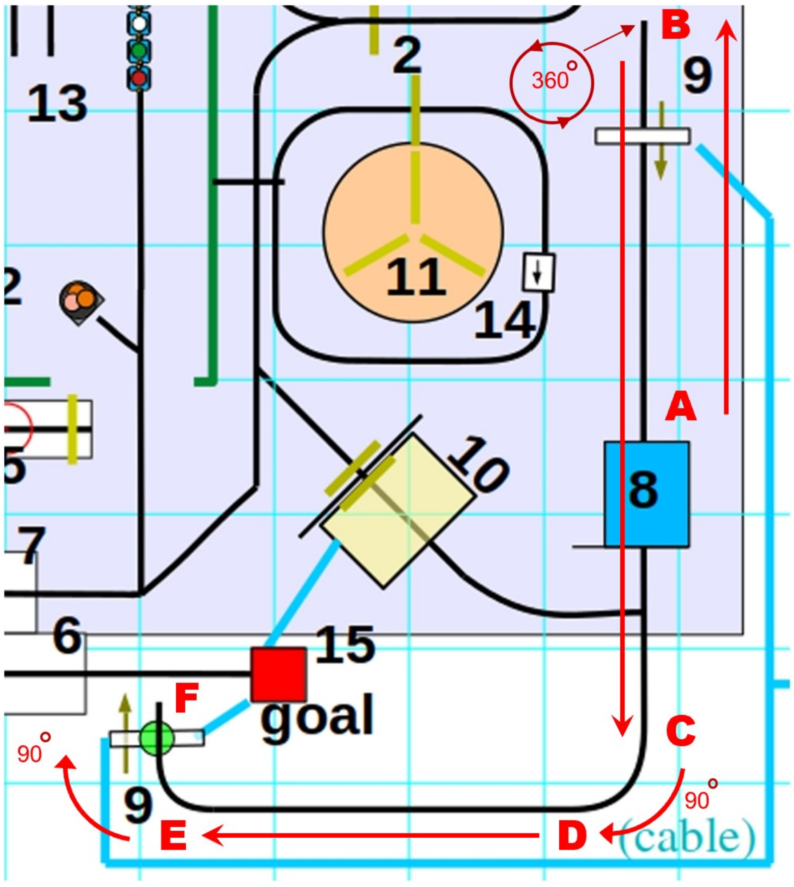


Figure 2.1

As shown in the figure 2.1:

The robot continue to follow this path until it reaches point B, which is near the starting point of the racetrack, and the robot will prepare to turn around on the spot, rotating itself towards the closed tunnel.

After that, the robot will keep following the line at a very slow speed of about 0.05 m/s as mentioned in the chapter 2.2 regarding the starting point posture issue. It will first correct its posture for a period of time until it passes through the start gate, where it will then accelerate to 1.2 m/s. This ensures that the robot's posture is zeroed and centered at the start, enhancing its stability.

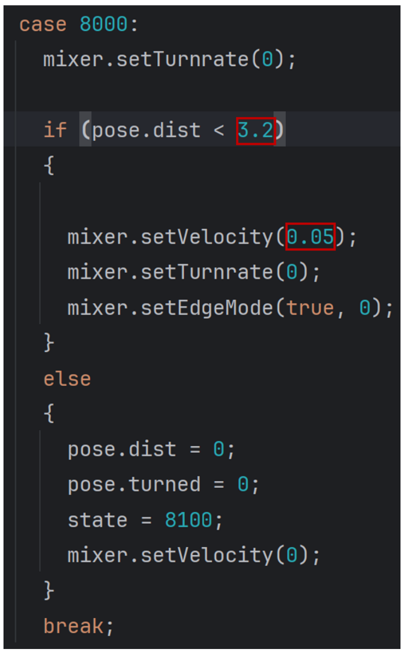
Then, the robot will continue to sprint straight to point C at a speed of 1.2 m/s.

As plotted in the figure 2.1 above. After reaching point C, the robot prepares to turn right until it arrives at point D. Upon leaving point D, the robot will sprint straight to point E. Finally, starting from point E, the robot will turn right and pass through the racetrack's finish gate at point F.

2.4 The solution

From point A to point B:

As shown in the figure 2.2, i use the accumulated distance recorded by the distance sensor to determine when the car reaches point B. When reaching point B, our robot stops.



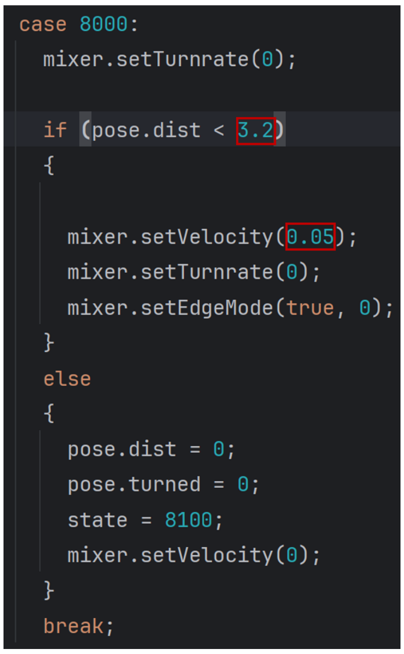
```
case 8000:
  mixer.setTurnrate(0);

  if (pose.dist < 3.2)
  {

    mixer.setVelocity(0.05);
    mixer.setTurnrate(0);
    mixer.setEdgeMode(true, 0);
  }
  else
  {
    pose.dist = 0;
    pose.turned = 0;
    state = 8100;
    mixer.setVelocity(0);
  }
  break;
```

Figure 2.2

Then, as stated in the figure 2.3, Turn 360 degrees on the spot to adjust the direction of the our robot. This state ends.



```
case 8000:
  mixer.setTurnrate(0);

  if (pose.dist < 3.2)
  {

    mixer.setVelocity(0.05);
    mixer.setTurnrate(0);
    mixer.setEdgeMode(true, 0);
  }
  else
  {
    pose.dist = 0;
    pose.turned = 0;
    state = 8100;
    mixer.setVelocity(0);
  }
  break;
```

Figure 2.3

From point B to point C:

As shown in the figure 2.4,There is a distance from point B to the starting gate, and i want the car's posture to be correct as it passes the start. Therefore, i have implemented logic for our robot to follow the line at a speed of 0.05 m/s, ensuring it stays centered on the line until it passes through the starting gate.

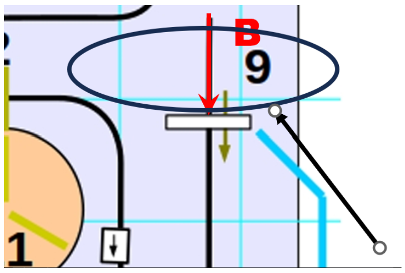
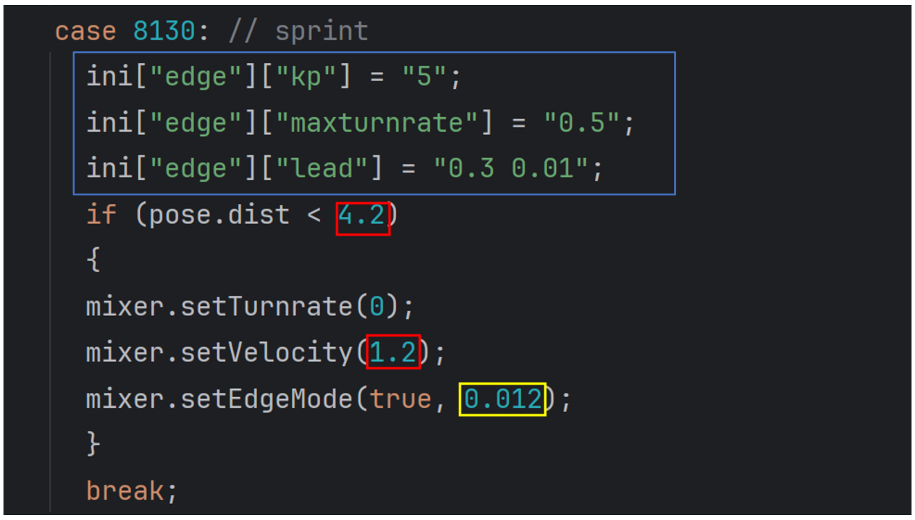


Figure 2.4



```
case 8130: // sprint
    ini["edge"]["kp"] = "5";
    ini["edge"]["maxturnrate"] = "0.5";
    ini["edge"]["lead"] = "0.3 0.01";
    if (pose.dist < 4.2)
    {
    mixer.setTurnrate(0);
    mixer.setVelocity(1.2);
    mixer.setEdgeMode(true, 0.012);
    }
    break;
```

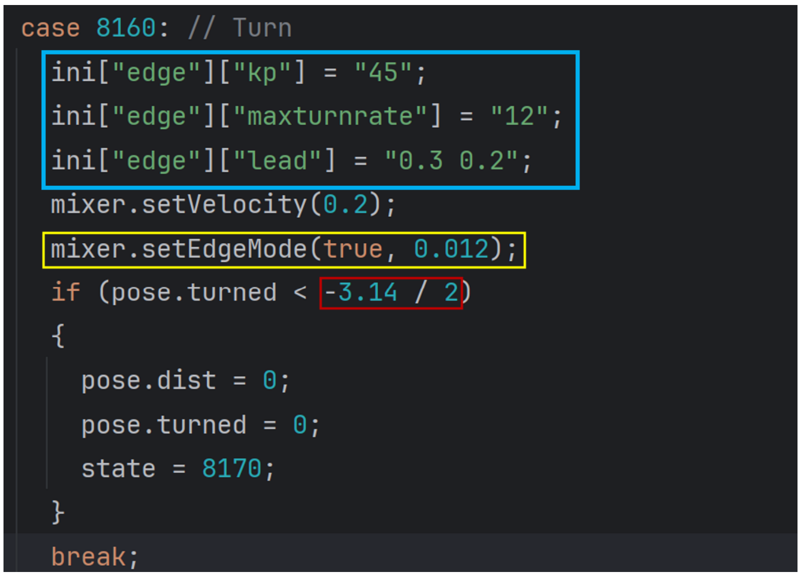
Figure 2.5

Then, the robot continuously sprints at a speed of 1.2 m/s to point C.

In the figure 2.5, you can see that we have set a different set of PID parameters for the sprint on the straight line. We use a two-dimensional array ***ini*** to instantly modify each parameter. I've lowered the ***maxturnrate*** to 0.5, a parameter closely related to maximum turning, as we don't want any corrections made on the straight line. I also reduced ***kp*** to 5 to prevent the robot from correcting and swaying. Finally, the two lead parameters are changed to 0.3 and 0.01, making the robot less sensitive to turning (though not as effective as the first two settings).

Additionally, I've added some offsets to the line tracking to increase tolerance, ensuring it stays in the middle of the line, rather than at the edge.

From point C to point D:



```
case 8160: // Turn
  ini["edge"]["kp"] = "45";
  ini["edge"]["maxturnrate"] = "12";
  ini["edge"]["lead"] = "0.3 0.2";
  mixer.setVelocity(0.2);
  mixer.setEdgeMode(true, 0.012);
  if (pose.turned < -3.14 / 2)
  {
    pose.dist = 0;
    pose.turned = 0;
    state = 8170;
  }
  break;
```

Figure 2.6

As shown in the figure 2.6, I significantly increased the values of ***kp*** and ***maxturnrate***, and reduced the speed to 0.2 m/s, allowing the robot time to react and making it highly sensitive to turning. Additionally, I reverted the ***lead*** values back to the default of 0.3 and 0.2 to allow for some correction.

From point C to point D:

Similar to the settings from point A to B, I adjusted the robot to have almost no sensitivity to corrections, ensuring it travels straight at a high speed. Then, I used the accumulated distance from the distance sensor to determine if the segment is complete, allowing for a transition to the next state.

From point D to point E:

Similar to the parameters shown from point B to C, I made the robot very sensitive to corrections and increased the turning angle while reducing the speed to allow it to navigate sharp turns. Then, I used the accumulated angle distance from the angle sensor to determine if the segment is complete, allowing for a transition to the next state.

From point E to point F:

Recover the default line-tracking parameters and The robot follows along a straight line and passes through the finish gate. The whole challenge is finished and the ideal complete time is 9.6 seconds.

2.5 Performance analysis

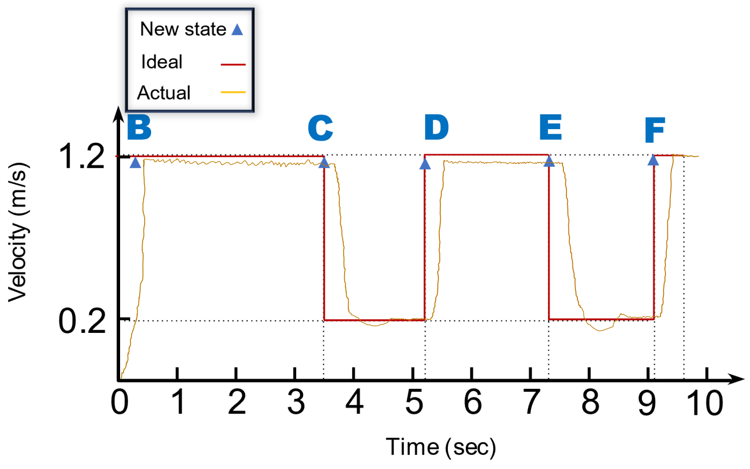


Figure 2.7: Comparison of actual conditions with ideal one: a graph of the relationship between velocity and time

Using data returned from logs, i can use mathlab to plot the actual relationship between speed and time, as shown in the figure 2.7. This graph illustrates the relationship between speed and time under ideal and actual conditions.

In the figure 2.7, i observe that the robot quickly accelerates to my set straight-line speed, having minimal impact on the total time. Additionally, i notice that the actual speed is slightly lower than the set speed (with an error within 3%, which is an acceptable range). This could be due to the sandpaper we applied to the wheels, which increased the rolling resistance, and the car is significantly heavier than the initial unmodified version due to the addition of a front roller and counterweights for balance. From point B to C and from D to E, the speed remains consistently stable. There are minor fluctuations, but these are nearly negligible, indicating high control stability. At the two turning points, performance is somewhat reduced; the robot's speed undergoes an increase and then a decrease in adjustment before stabilizing. This might cause the robot to enter the turns too quickly, failing to turn in time, and losing the line in the end.

The figure 2.8 below is an ideal trajectory map of the robot completing the entire racetrack, created using MATLAB. I have also processed the data to zero the starting point's x-coordinate for easier analysis. By using log data returned from the sensors during the race, we can plot the actual trajectory curve, which allows us to visually compare the robot's actual performance against the ideal trajectory.

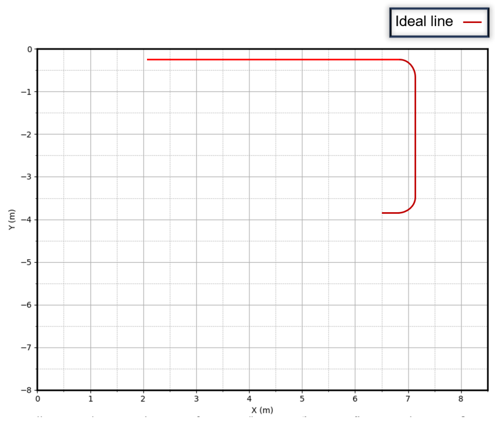


Figure 2.8: The ideal trajectory of racetrack

As presented in figure 2.9, In both runs, the robot successfully completed the entire course. However, during the first attempt, there was excessive ineffective correction and swaying on the straight sections. Notably, during both turns, the robot made significant corrections, frequently needing to reposition itself, which resulted in time loss.

In conclusion, the robot performed well on the racetrack, but it lacked stability, which will be discussed in detail in the following sections. This is why we in the end have to reduce the speed on straight sections, sacrificing speed to gain stability and reliability.

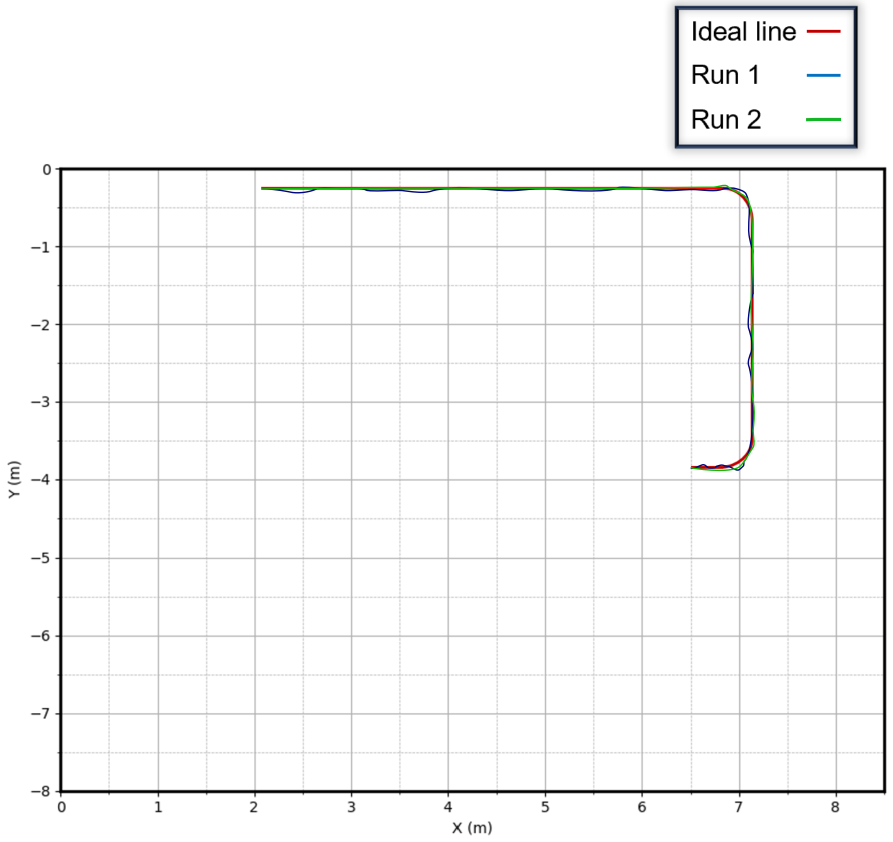


Figure 2.9: Comparison of two actual trajectory with ideal line

2.6 Possible accidents that might occur

In some cases, after sprinting on straight sections, the robot may fail to turn in time, causing it to run off the track.It can be seen in the figure 2.10.

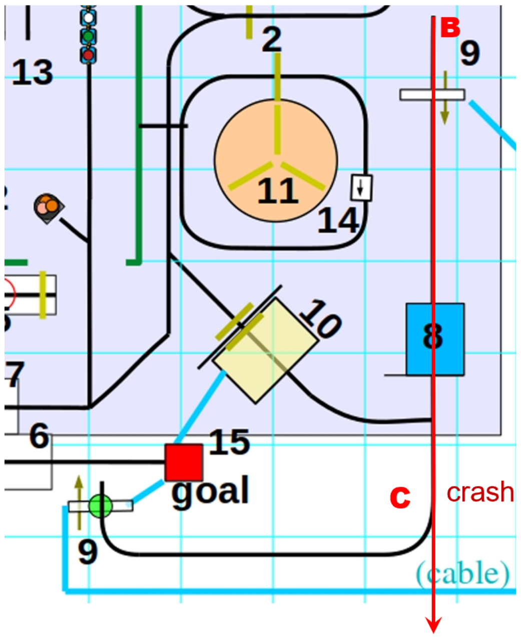


Figure 2.10: Failure scenario

Furthermore, after repeated testing, this type of failure did not occur in other parts, indicating that the failure is not a rare random event, but rather a consistent error with a stable probability of occurrence. This is likely due to shortcomings at the code level (PID parameters or logic).

Last but not least, in rare cases due to poor weight distribution with the center of gravity shifted towards the rear, the robot may tip forward, resulting in significant lateral swaying on straight sections.

2.7 Reliability

The very rare random failures mentioned before can be disregarded for now, as their occurrence is exceedingly rare. Our primary focus should be on failures that occur consistently - a failure where the robot is unable to turn promptly , causing it to run off the track from a straight section.

Consequently, I did a statistical analysis: in 50 consecutive tests, the robot failed 20 times and successfully turn around the corner 30 times.

Then, the failure rate can be approximated using the following formula:

$$\lambda(t) \approx \frac{\text{Number of Failures}}{\text{Total Number of Tests}} \tag{2.1}$$

So, the failure rate would be approximately:

$$\lambda(x) \approx \frac{20}{50} = 0.4 \tag{2.2}$$

Also, I utilized the failure rate data to plot a failure density function graph, which effectively illustrates the trend of failure rates over time. It is evident that the failure rate continuously increases, reaching a peak around 40%.(As shown in the figure 2.11)

This observation also highlights that, with changes in track conditions and increasing amounts of dust, the issue becomes more pronounced as the sandpaper surface of the tires wears down, reducing friction.

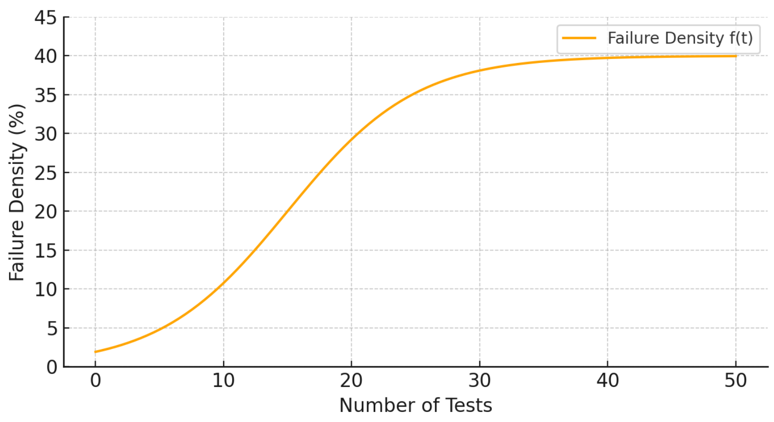


Figure 2.11: Failure density function

2.8 Possible improvements

First□continued adjustments of the PID parameters during turning were made to enhance the robot's sensitivity to turns and increase the turning angle. For example, adjustments included increasing the sampling frequency, raising the kp value, and setting a higher maximum turn rate. Also, modifying the offset during line tracking to position the robot closer to the center of the curve was implemented to prevent it from veering off course.

Secondly, the auxiliary front wheel was moved forward, and additional counterweights were designed to be added to the front, balancing the weight and shifting the center of gravity forward. As a result, the robot can sprint and turn more quickly and stably, effectively preventing uncontrollable swaying and nose lifting under significant torque.

Finally, it may be possible to adjust the motor parameters to reduce the torque output during turns, thus avoiding wheel spin and slippage. This approach might have improved the reliability, but it was not tested.

2.9 Conclusion

The solution to the challenge appeared ingeniously designed and performed impressively in terms of speed. However, due to a lack of robust reliability, frequent errors occurred, and the methods to address failures were complex. Unwilling to take risks, we ultimately reduced the speed on straight sections to enhance the challenge's stability.

A

Matlab data plot example

A.1 XY-plot code

Here is the XY-plot code in the chapter 2.5.

The saved data is ***not*** from the competition track but on the ASTA.The conclusion may be biased

```matlab
% Load data if necessary
dd = load('log_pose_abs_1.txt');
dd2 = load('log_pose_abs_2.txt');
% Create figure
h = figure(120);
clf(h); % Clear figure
% Plot data from first dataset
plot(dd(:,8), dd(:,9), 'LineWidth', 2); % Adjust line width as needed
hold on;
% Plot data from second dataset
plot(dd2(:,8), dd2(:,9), 'LineWidth', 2); % Adjust line width as needed
% Customize axes and grid
ax = gca;
ax.XLim = [0, 6];
ax.YLim = [-1, 0];
ax.XTick = 0:0.5:6; % Major and minor ticks combined
ax.YTick = -1:0.1:0;
ax.GridLineStyle = '-';
ax.MinorGridLineStyle = '--';
ax.FontSize = 14;
grid on;
grid minor; % Show minor grid lines
% Labels and titles
xlabel('X (m)');
ylabel('Y (m)');
% Equal scaling for both axes
axis equal;
% Legend
legend('Run 1', 'Run 2');
% Add a solid border around the plot
for k = ["top", "bottom", "left", "right"]
    ax.(k).LineWidth = 2; % Set border line width
    ax.(k).Color = 'black'; % Set border color
```

```
end
% Save the plot
saveas(h, 'challenge_xy.png');
```

Listing A.1: Plotting of two runs to evaluate the repeatability of the challenge.

The plot from this list is shown in figure 2.9.