HIGH SCHOOL EDITION
Journal of Student Research
Volume 11 Issue 2 (2022)

Strategy Optimization in a Robot Race Using PID

Song Yue David Li[1] and Jie Li[#]

[1]Westlake High School, Austin, TX, USA
[#]Advisor

ABSTRACT

In this paper, the proportional-integral-derivative (PID) Controller is optimized to complete a robot challenge, *The Race*, by the University of Texas at Austin's Robotics Academy. Optimization is done through three stages: (A) optimizing the PID coefficients of both wheels; (B) optimizing the constant speed, and (C) setting the constant speed to its maximum. Finally, the analysis proves that Method C, which yields the fastest time, approaches the theoretical bound.

Introduction

In 2021, the University of Texas at Austin held a Robotics Academy [1] for high-school students that taught the basics of C++, Robot Operating System (ROS), and robot movement. There was a sequence of tasks that needed to be completed before reaching the final challenge called *The Race*.

The Race, meant to show attendees' knowledge of C++ and understanding of robot automation, consisted of a loop that had turns and straight lines. When launching the racetrack on ROS, a timer would also appear, and when started, would begin moving the robot. The challenge was to navigate the robot on the line without letting it drift off. The time would appear once it finished a complete lap. The best time that the academy had recorded on simulation was 20 seconds, but the time they wanted the students to achieve was anywhere under 30 seconds.

This paper discusses how to optimize the algorithm, tactics, and parameters of the proportional-integral-derivative (PID) Controller [2], to achieve the fastest possible time. With a limited number of attempts, the constant speed (S_c), proportion (K_p), derivative (K_d) and integral (K_i) were developed to achieve the target of the fastest time in one lap.

The organization of this paper is as follows. The model is introduced in Section II, optimization process in Section III, analysis in Section IV, and conclusion in Section V.

Model Description

When ROS launches the racetrack from the Texas Robotics Academy repository, this circuit map of *the Race* is shown like in Figure 1.

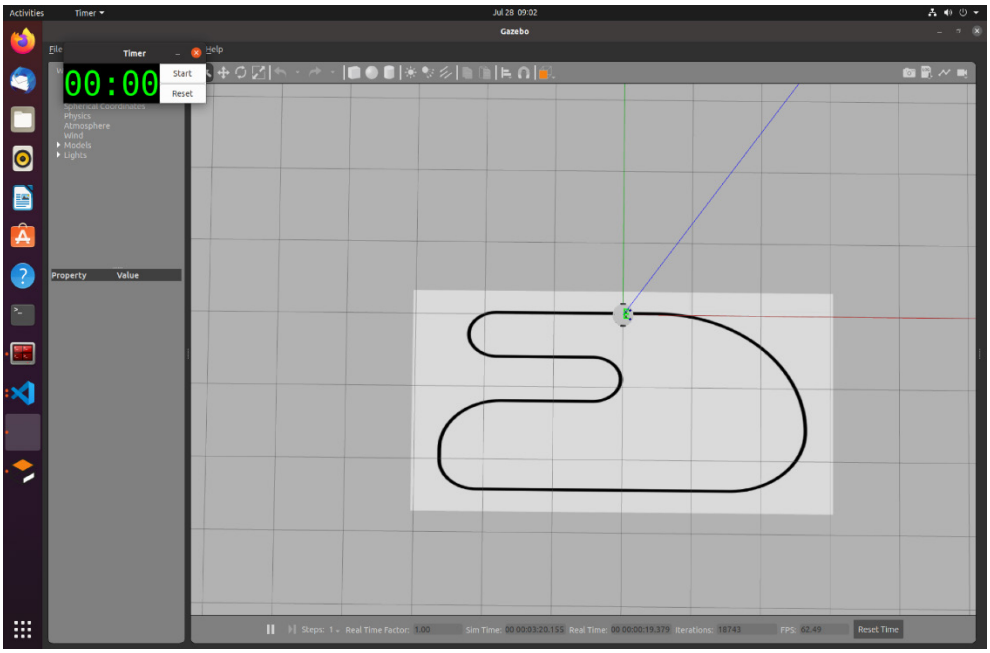

Figure 1. Circuit map of the Race.

As shown above, the robot starts at the center of the simulation and must travel around the track. The time will be recorded once the robot returns to its starting point. The only tools that are given to the robot are an array of line detecting sensors. There are a total of eight sensors and these sensors output a number in respect to if the line is beneath the sensor, as shown in Figure 2.

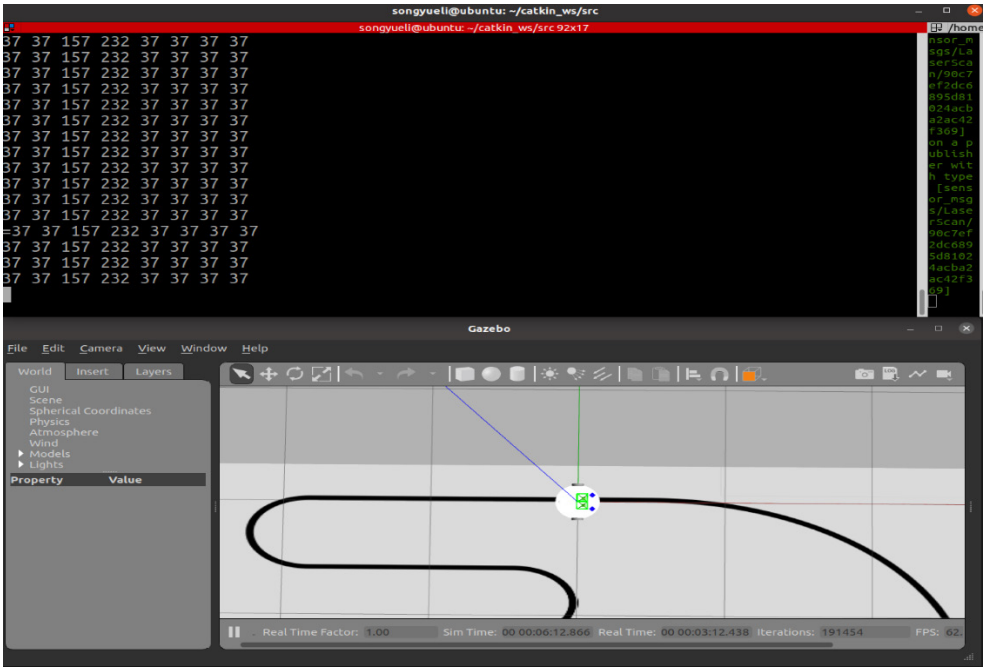

Figure 2. Analyzing the outputs from line sensors.

At the starting position, the 4th sensor on the robot outputs the highest value, and when the robot is moved around, the highest number will change between the sensors. Using this information, the location of the

robot relative to the line can be found, in an index of 0 to 7. For the challenge, the index needs to be changed so that it is within -1 and 1. This can be done through the function, *f(index) = (index - 3.5) / 3.5.*

Therefore, if the robot has drifted all the way to the right, then the final output will be -1, and when all the way to the left, will output a 1. Shown in Figure 3.

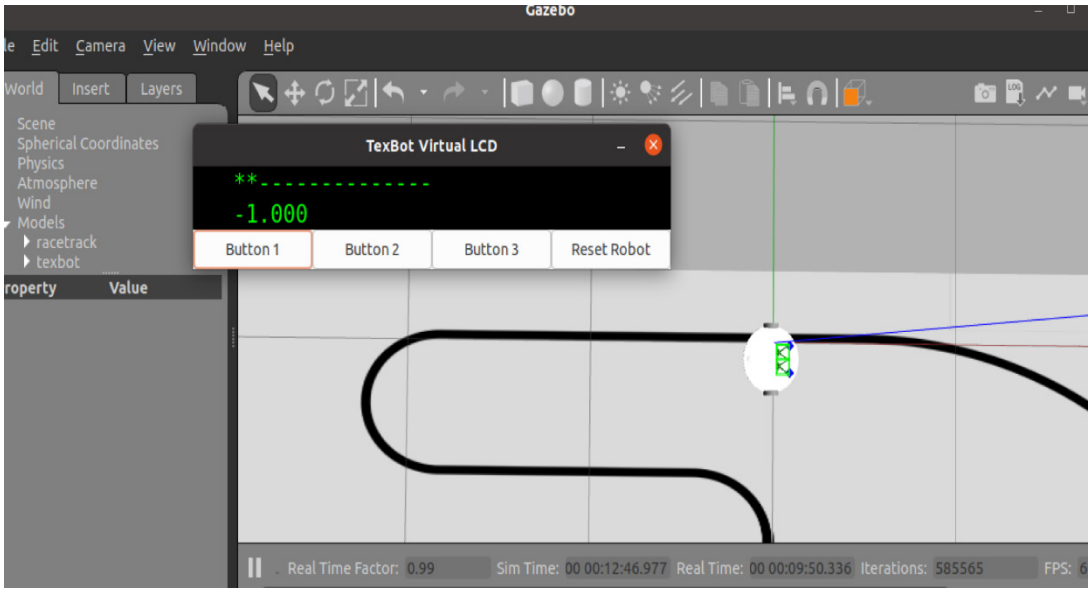

Figure 3. Scaling the output.

The number on the second line of this LCD screen represents the threshold used to pick out the largest number in the array of outputs of the sensors, as shown in Figure 4.

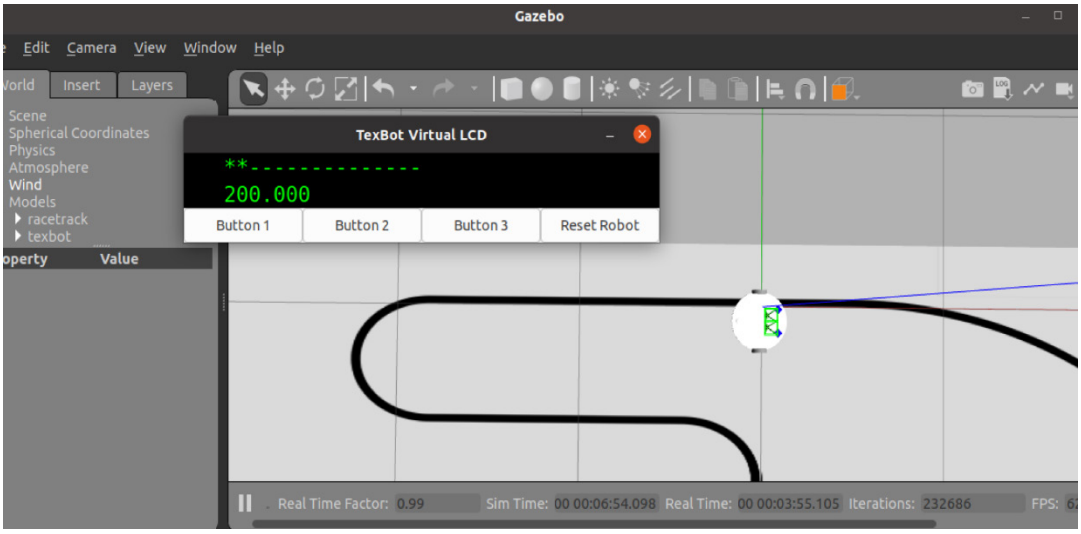

Figure 4. Finding the threshold to identify the location of the line.

The goal of this challenge is to make it completely automated. The way the robot moves is by giving the left and right wheel a speed within -100 and 100. Exceeding the value will result in an error and the robot will cap the speed at 100.

To call the robot to move, the following declaration should be used:

bot.move(leftSpeed, rightSpeed);

If the speed of the left wheel is the same as the right, then the robot will move forward at a constant speed. Similarly, if the right moves faster than the left, or vice versa, the robot will turn in an arc. Only when the wheel speed of one side is the exact negative of the other, will the robot turn in place. An example is shown in Table 1.

Table 1. Example of robot movement as a function of left and right wheel speeds.

Left Wheel Speed	Right Wheel Speed	Robot Movement
100	100	Forward at full speed
-100	-100	Backward at full speed
100	0	Turning to the right, while creating the smallest arc
0	100	Turning to the left, while creating the smallest arc
100	-100	Turning in place to the right at full speed
-100	100	Turning to the left at full speed

A PID loop was intended to autonomously move the robot across the track. The declaration looks like this:

bot.move(S_c + (numP * K_p+ numI * K_i+ numD * K_d),
S_c - (numP * K_p+ numI * K_i + numD * K_d));

where

numP is the error from the robot's current position;
numI is the total amount of accumulated error;
numD is the difference between the previous and current error, and
K_p, K_i, and K_d are coefficients that need to be manually adjusted. S_c represents the constant speed that the robot will move at, and the PID term is either added or subtracted to S_c so that the robot turns on the track.

Optimization Process

Three methods are presented to improve the PID coefficients step by step.

A. Optimize the PID Value to Both Left and Right Wheels

Starting with S_c = 50, the robot moves very slowly, and is therefore very easy to calculate K_p, K_i, and K_d.
To get the PID coefficients, it is best to set K_i and K_d to 0 and adjust K_p until the robot moves without undulation. Using 50, 25, 0.001, and 5 for S_c, K_p, K_i, and K_d, respectively, the robot successfully moves along the track with ease. However, these numbers resulted in the time of 36 seconds, which did not get under the goal time of the challenge. The key factor limiting the speed is S_c, so the next step is to increase S_c, and repeat the steps to find K_p, K_i, and K_d. At S_c = 65, the fastest time possible is around 27 seconds and when S_c = 68, approximately 26 seconds. However, further increasing S_c would cause the robot to drift off the track.
A huge factor affecting the performance is the use of K_i. K_i is important in keeping the robot smooth during straight lines. Since there are 8 sensors on the robot, and they're scaled to be between -1 and 1, the value of numP can never be 0, which is the ideal output when the robot is exactly centered on the line. The array of numP looks like this:
-1.00, -0.71, -0.42, -0.14, 0.14, 0.42, 0.71, 1.00

This means that when the robot is moving on a straight line, numP toggles between -0.14 and 0.14, which is a major obstacle to a fast time. The integral aspect of PID and K_i help to smooth the trajectory of the robot and improve the time.

Although this is an impressive feat, but the fastest time recorded was 20 seconds. So, there were further improvements needed to be made.

B. Optimize the Constant Speed, $\mathbf{S_c}$

The problem with Method A is that there is too much momentum when the robot hits a curve, causing it to slide off the track. To compensate for the momentum, one would naturally increase K_p, but this does not work. That is because the speeds for the left and right wheels cannot exceed 100, which becomes a problem when the robot is on the peak of the curve and num is equal to -1 or 1. This is shown in Figure 5.

Figure 5. Graph of S_c in relation to K_p.

Given the formulas of the left and right wheel speed as
S_c+ (numP * K_p + numI * K_i + numD * K_d), and
S_c - (numP * K_p + numI * K_i + numD * K_d), respectively,
if K_p+ S_c > 100, ROS reports an error, and the robot does not move as intended. This explains why the maximum S_c that works for the method A is around 68 to 70. To compensate for this limitation, S_c should also be changed according to numP so that K_p is not the only factor changing the wheel speeds. Since there are absolute values of numP, ignoring the negative or positive, four different values of S_c should be used, relative to its position on the line. The improved algorithm is indicated as following pseudo-code:

```
if      (numP < 0.2 && numP > -0.2) { Sc =75; Kp=28; Kd=12; Ki=0.002;}
else if (numP < 0.5 && numP > -0.5) { Sc =71; Kp=34; Kd=17; Ki=0.001;}
else if (numP < 1   && numP > -1)   { Sc =65; Kp=36; Kd=20; Ki=0.001;}
else                                { Sc =60; Kp=36; Kd=24; Ki=0.000;}
```

S_c decreases as the |numP|>>0, which allows the robot to slow down on turns, increasing stability and consistency. Another important observation is that K_p needs to be changed exponentially, while the S_c and K_d need to be changed linearly. K_p is changed exponentially to allow the robot to make smoother turns. Otherwise, the behavior of the robot seems shaky, and it flies off the track.

Like Method A, increasing S_c at each threshold decreases the accuracy. The time achieved with this method is 25 seconds, and sometimes even 24 seconds.

C. Optimize by Setting $\mathbf{S_c} = 100$

The previous two methods have achieved impressive time, but they are still far from the fastest record. The problem is that the speed of the left and right wheel revolves around S_c; one increases and another decreases. In fact, the robot can turn as long as the two wheels have different speed. To achieve the fastest time, one wheel must be 100**.** Therefore, the improved algorithm is as follows:

```
if          (right turn)
bot.move(100, 100 - (numP * Kp + numI * Ki + numD * Kd));
else if (left turn)
bot.move(100 + (numP * Kp+ numI * Ki + numD * Kd), 100));
```

While K_p, K_i, and K_d are fixed during the entire race, it would be more efficient to integrate the if-else statement from Method B to this method. This allows joint optimization for increasing consistency and stability on turns. The final optimal algorithm is as follows:

```
Let pid = numP * Kp+ numI * Ki + numD * Kd;
if (numP < 0.2 && numP > -0.2) {Kp=30;   Kd=10; Ki =0.002;
if(numP > 0) bot.move(100, 100 - pid);
else        bot.move(100 + pid, 100); }
if (numP < 0.5 && numP > -0.5) { Kp=42.5; Kd=13; Ki =0.001;
if(numP > 0) bot.move(100, 100 - pid);
else        bot.move(100 + pid, 100); }
if (numP < 1 && numP > -1) {Kp=55.5; Kd=15; Ki =0.001;
if(numP > 0) bot.move(100, 100 - pid);
else                    bot.move(100 + pid, 100); }
else    {Kp=71;   Kd=18; Ki =0.000;
if(numP > 0) bot.move(100, 100 - pid);
else        bot.move(100 + pid, 100); }
```

Notice that *if* statements are used rather than *if-else* statements. By doing this, the turns are smoother because the K_p values are gradually changed within each while loop.

Finally, the record of 20 seconds was achieved. However, as S_c increases, there are more chances that the robot is out of the track. Through this observation, it makes sense that the fastest time of 20 seconds is achieved around 10% of the experiments, 21 seconds is achieved around 20% of the experiments, and the other experiments are incomplete. The problem is caused by the even number of sensors which makes numP=0 impossible, along with inconsistencies in the simulation.

Results and Discussion

The distance and speed are obtained through the simulation to calculate time. Assume the track is scaled so that each grid square is exactly one square inch, then the total distance of the track, along with the speed of the robot can be measured as shown in Figure 6, while assuming that the turns are arcs.

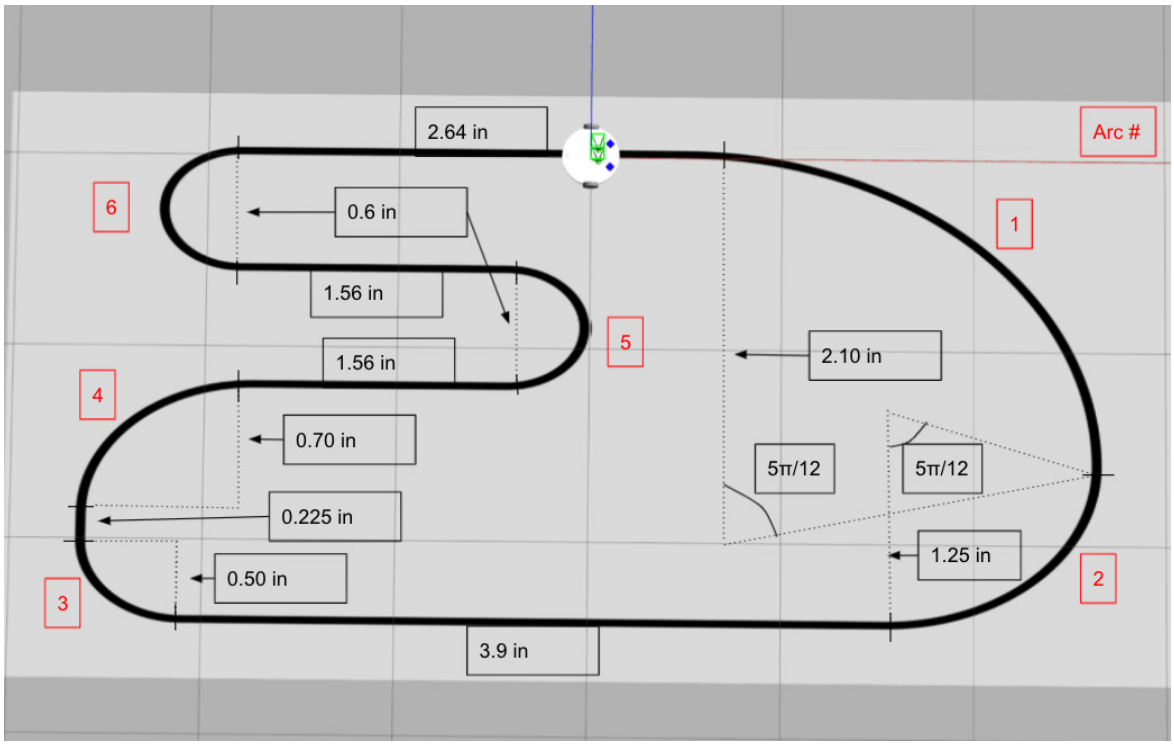

Figure 6. Calculation for optimal time.

It is measured that the robot travels about "***1 inch in 1.0 seconds***" when the power of the left and right wheels is at the max speed of 100, excluding its acceleration. There is a total of approximately 17.97 inches of track. If the track is unraveled into a straight line, and the wheels maintain a constant speed, the robot would take approximately 18.04 seconds. However, the fastest time achieved in the experiment is 20 seconds. This is because multiple turns on the track force one of the wheels to slow down and therefore it takes more time to travel the arc than to travel the straight line.

To calculate the optimal time, T_{opt}, with the turns (refer to Equation 1), the arc length, $\hat{L}$, needs to be found by measuring the radius and arc degrees, and the relative speed, $\overline{V}$, is averaged from the left and right wheels.

$$T_{opt} = \hat{L}(in) \times 1(^{sec}/_{in}) \times \frac{100}{\bar{V}} \qquad (1)$$

For example, on arc 1, when the left and right wheel speeds are 100 and 83, it is measured that the robot completes a 75° arc, with a radius of 2.10 inches, so $\hat{L} = 2.75\ in$, and $\bar{V} = 91.5$. After substituting into (1), $T_{opt} = 3.0\ sec$. The estimated optimal time on all arcs are shown in Table 2.

Table 2. Estimated optimal time on arcs.

Arc Index	Radius (Inch)	Speed (Left/Right)	Arc Degree	T_{opt} (sec)
1	2.10	100/83	75°	3.00
2	1.25	100/75	75°	1.87
3	0.50	100/45	90°	1.08
4	0.70	100/59	90°	1.38
5&6	0.30	100/29	180°	1.46

Therefore, the total time on the arcs is 10.25 seconds. Adding the time for the straight lines (9.89 seconds), the total estimated time of the track is 20.14 seconds. This number is treated as the theoretical (calculated) bound of the optimal (fastest) time.

The above analysis proves that the record time is close the optimal time, and that the optimal methods we proposed for PID controller yields the fastest results in *The Race* game.

Conclusion

Through lots of experiments and tuning, the goal of getting the record time of *The Race* game has been achieved through optimizing the coefficients of the PID controller to achieve the fastest speed. The method is to adjust only one of the wheels to account for turns. The coefficients for the PID controller, especially K_i, have been specifically optimized step by step to improve the performance. The experiments and analysis prove that the fastest time of *The Race* game is about 20 seconds, and this is the fastest time possible to achieve.

Researchers not limited to robotics, who need to fine-tune the PID coefficients to optimize their performance, can refer to the methods and steps presented in this paper.

Acknowledgments

I would like to express my gratitude to the University of Texas at Austin for their excellent robotics camp and the student counselors who guided me through the project. I would like to express my upmost appreciation to Dr. Jie Li, who encouraged my research and taught me PID. His insight and collaboration offered this project the best results. I would also like to thank my entire family, including my dog, Ryzen, for their sponsorship and encouragement over the project.

References

[1] University of Texas at Austin Robotics Academy 2021, https://www.cs.utexas.edu/outreach/academies
[2] PID Controller, Wikipedia, https://en.wikipedia.org/wiki/PID_controller