AE 4610 Project

SANIC: PID Line following robot using IR sensors

By: Noe Lepez Da Silva Duarte

Spring Semester 2022

I. Introduction

The following project explores creating and controlling a self-driving, line-following robot made using the Webots software. Webots is an open-source robot simulator that provides a complete development environment to model, program, and simulate robots [1]. This project focused first on modeling a simple robot, affectionally called SANIC, shown below. The lines visible on the robot's front side represent Infra-Red (IR) sensors used to determine its position about the line it must follow.

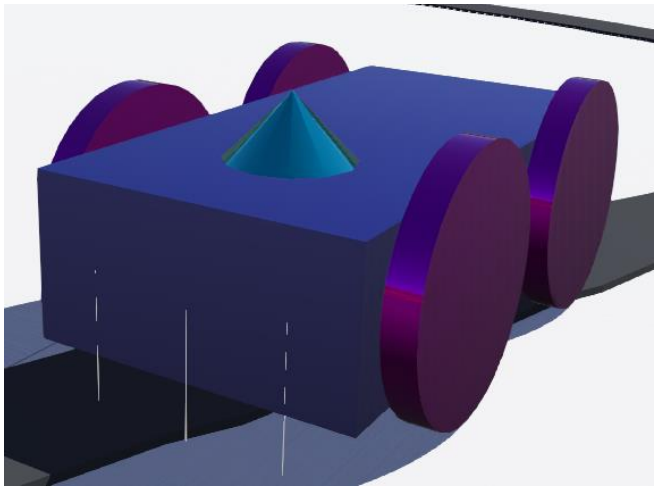

Figure 1. SANIC model

IR sensors work by sending an infra-red signal and measuring the amount of IR light it receives back. As less light is reflected by darker colors (pure black absorbs almost all light) [2], readings from an IR sensor can be used to guide the robot and ensure it stays on the line.

Once SANIC was modeled, a preliminary environment was set up. Its controller was created, first using a simple controller logic to turn if it was off the black line and progressively improved to add more intelligent logic. Eventually, a PID controller was coded into the robot to ensure its speed would change according to the position of the line and the turn angle. Multiple gain values were tested to improve SANIC's ability to take shaper turns faster and ensure it always stays on track.

Finally, SANIC's environment was created with numerous tracks to replicate real racing tracks and test the robot's abilities and limitations. This project could then be used to build and test a real robot.

II. Robot Design

While Webots provides numerous pre-made robots, creating a robot seemed to be useful in better understanding the process behind creating real-life robots and helped apprehend how to create a controller.

Firstly, a robot was created using the "Robot" base node available in the software. The body was made as a 0.2x0.1x0.05m box, which could house any electronics and mechanisms to make SANIC function properly. Four hinge joints were then created. Hinge joints connect the robot's body and the motor connected to the wheel, as pictured in Fig. 2.

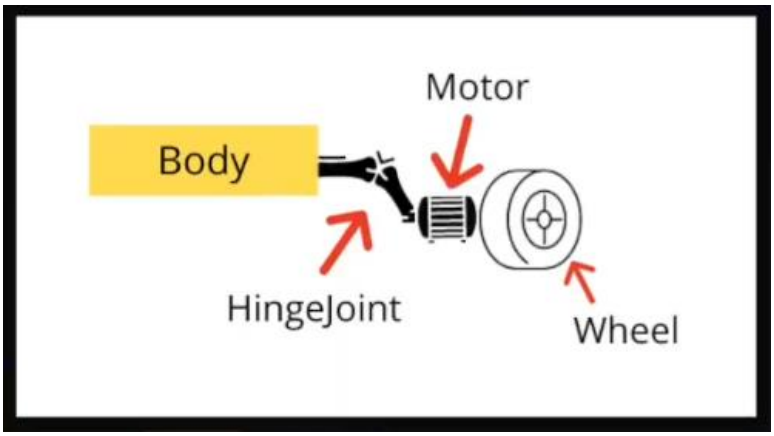

Figure 2. Hinge joint diagram [3]

In Webots, hinge joints have three fields that can be modified. Firstly, the "jointParameters" were updated to make them rotate about the y-axis. A rotational motor was then added to the "device" field to allow it to move the wheels. Finally, the "endpoint" was made of a solid cylinder of height 0.01m and a radius of 0.04m. Their centers were translated to the vehicle's four sides, at (±0.05, ±0.055) m on the robot's body. The joint anchor points were also updated to match this to allow proper rotation of the wheels. These four separate joints would allow for a differential drive.

The simulation was then started to ensure that the robot was stable when not moving. However, an issue arose. The wheels went through the map environment due to the objects not having bounding environments or physics. Indeed, while objects may be created within Webots, they cannot have physical interactions unless a bounding object is made to surround them. Bounding objects were created for the body and each wheel, and physics was enabled.

Next, three "DistanceSensors" were created, with infra-red type to allow the robot to 'see' the ground, especially the differences in color as it moves around. One was placed at the face's center, and the two others were equally spaced to the right and left by 0.03m. The aim behind having three sensors was to have the outer ones on the ground while the central one stays looking at the track

III. Robot Controller Design

Preliminary environment creation

The "RectangleArena" was first increased to a 5x5m arena to allow SANIC to have more space to move. The floor color was also made to be white to have higher contrast and difference in IR readings between the white floor and black track. A pre-made track was also imported [5].

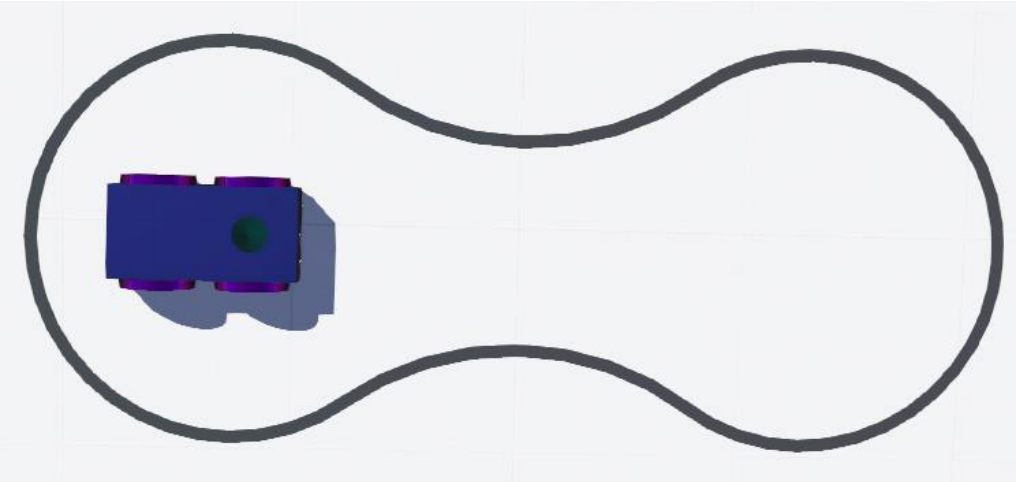

Figure 3. Preliminary track

Initial Controller design

Due to my lack of prior experience with Webots, the Webots documentation, numerous models, and videos were used to create the PID controller. These are cited as [6] to [9].

A Python controller was made for SANIC. The robot class was first imported from the "controller" library to represent and control the robot using the code. Each motor and sensor were imported and given an appropriate name. This allowed to quickly read the IR sensor's values and change each motor's speed.

The main loop was then made as a while loop, which would only end once the timestep was equal to -1. The timestep represents how long the simulation has been running and equals -1 only when it is reset (it can be paused and accelerated at will). SANIC was instructed to go forward while IR readings were printed in the console. While on the white ground, readings did not exceed 600. However, IR readings peaked when the robot crossed the track, with values reaching the threshold of 1000, implying that any IR reader looking down on the track would have much higher values than if looking down on the white ground.

With its combination of three IR sensors, the robot could encounter four different situations:

1. The central sensor could be on track (with a value over 700), with the two outer sensors on the white ground (with values below 700). In this case, the robot is on track and does not need to correct its trajectory.
2. The central sensor could still be on track, but one of the outer sensors could also find itself on the track, indicating the start of a turn. In this case, the robot needs to change its direction to follow the turn.
3. If the central sensor is off the track, with one outer sensor on the track, the robot encounters a sharp turn and must readjust its trajectory fast to stay on track.
4. All sensors may find themselves off the track in the case of a very sharp turn. If this is the case, the robot will continue its fast turn until it reaches the track again.

While this functioned well in the initial test, the sudden changes in wheel velocity made SANIC toss around as it had negligible mass. Once the mass was increased and the maximum velocity decreased, the robot could follow the preliminary track well. To ensure SANIC's ability to move around, a new, more challenging track was made to test out the robot's capabilities.

Additional Track Creation

A new track was made in TinkerCad [10]. TinkerCad is an online-based tool made to let users create 3D CAD models. The track's shape was made arbitrarily but needed to have sharper turns than the preliminary track to ensure SANIC's competence in different environments. The track was drawn using a mouse and the "scribble" option. It was made to be flat and of solid black color. It was then exported as a .obj file and imported into Webots. This new track is shown below with SANIC as a size reference.

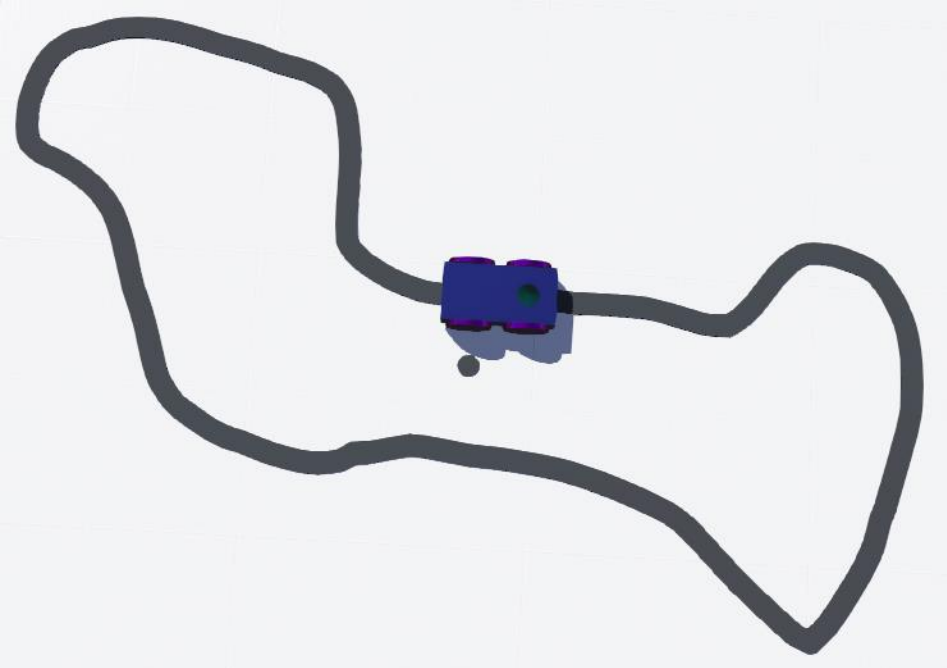

Figure 4. Track made using TinkerCad

PID Implementation

SANIC could not stay on track unless its maximum speed was significantly reduced. While this was a way to fix the robot, it made it move exceptionally slowly, impractical if it was to be reproduced in real life. The code was improved to add a PID controller to make the robot move faster and more smoothly.

Firstly, an error variable was created rather than changing the speed at every if statement. This error equaled 0 if only the middle sensor were on the line. It would also update to +1 or -1 if both the central and outer sensor were on the line and +2 or -2 if only the outer sensor was on track. The higher the absolute error was, the further away from the track the robot would be. The following relationships were then derived to determine the proportional, integral, and derivative error, respectively: P, I, and D.

$$P = error \quad (1)$$

$$I = error + I \quad (2)$$

$$D = error - last\_error \quad (3)$$

$$Balance = k_P P + k_D D + k_I I \quad (4)$$

Equation (4) shows the balance of errors, using the respective PID gains and values. These gains were derived using trial and error, and found to be $k_P = 1.0$, $k_I = 0.005$, $k_D = 1.9$. Here, the proportional term instructs the robot on how to drive around the curve, the integral term seeks to reduce errors throughout the system, and the derivative term seeks to always keep the robot on the road. Should any of these gains be too high, the robot would have erratic behavior, suddenly speeding up and slowing down, and have more trouble making sharp turns. On the other hand, low gain values would make the robot lack responsiveness and have trouble staying on track.

The balance term determines the overall error from the three controller components. The controller then uses the balance term to define new left and right motor speeds as,

$$Left\ speed = max\ speed - balance \quad (5)$$

$$Right\ speed = max\ speed + balance \quad (6)$$

Where the max speed is a speed pre-defined by the user (defined as three here), five new situations arise. If the left speed is greater than the max speed or the right speed is negative, the robot must turn right. The right motors will turn off, and the left motors will go to the speed calculated as the left speed. The opposite occurs if the right speed is greater than the max speed or the left speed is negative. Finally, if the right and left speeds are equal, the error in the system is 0 (or negligible), and the left and right motors move forward. By giving positive or negative error weights when an outer sensor is on track, the robot can quickly understand whether it must turn right or left and calculates a new speed to accomplish this as fast as possible.

5. Robot Testing

Once the controller was complete, the robot was tested on different tracks to investigate the extent of its abilities. Firstly, a copy of the Monte-Carlo track, Fig. 5, was created, and SANIC sped through the track in 1 minute 30 seconds and 744 ms. The main challenges here were the sharp turns at the top-right of the track and the sharp turn at the bottom. While the robot did slightly overshoot on these, it never got fully off track.

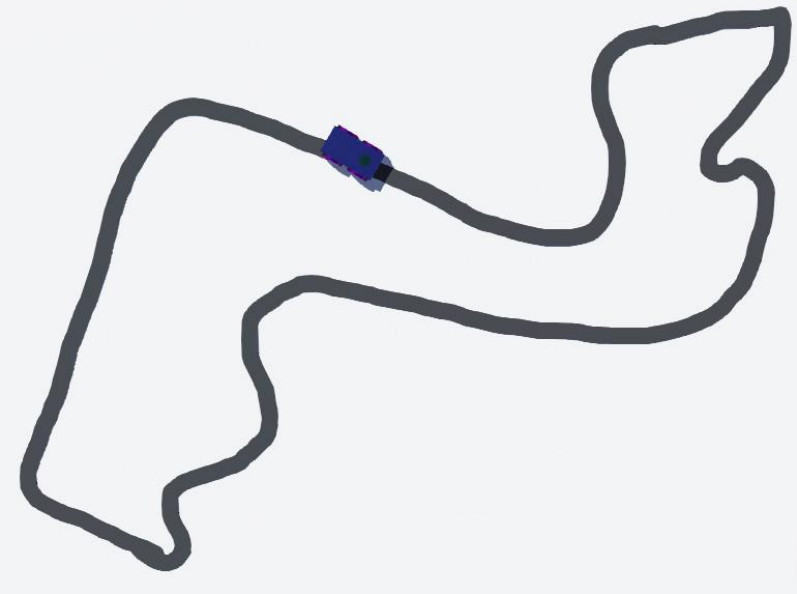

Figure 5. Monte-Carlo track made using TinkerCad

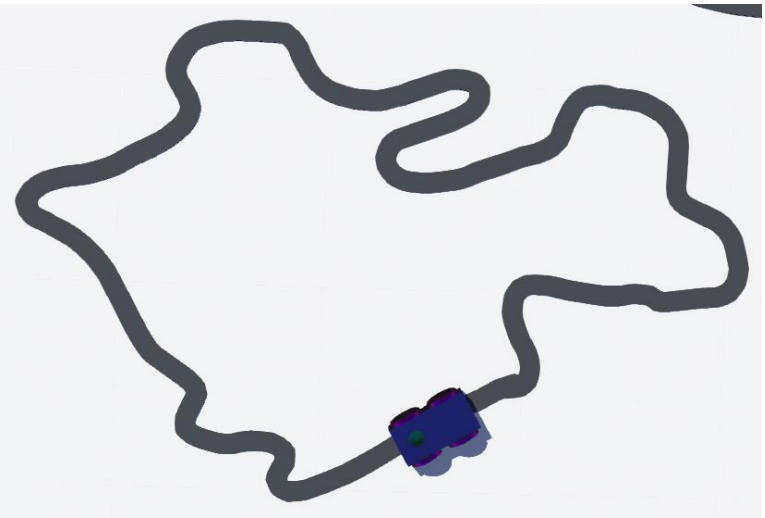

Figure 6. Nürburgring - Nordschleife track made using TinkerCad

Next, SANIC was tested on the Nürburgring - Nordschleife track, shown in Fig. 6. The robot was initially unable to make the large U-turn at the top of the track. The proportional derivative was

increased to attempt to remedy this. However, while it was able to go around the U-turn, SANIC could not follow the initial set of 4 turns. The gains were then reset back to their original values, and the speed lowered to 2, which allowed the robot to go around the track. It did have trouble throughout these regions, getting fully off-track on the sharp U-turns at the top of the track before rejoining the circuit. Nevertheless, it completed a lap in 1 minute 22 seconds and 120 ms.

Due to the robot's issues with sharper turns, it was tested on three other shapes to assess the extent of its abilities. Firstly, SANIC went around a rectangular track with rounded corners, shown in Fig. 7. It had no issues doing so, even with a speed of 3. This was expected as it had no issues with such turns in previous tracks.

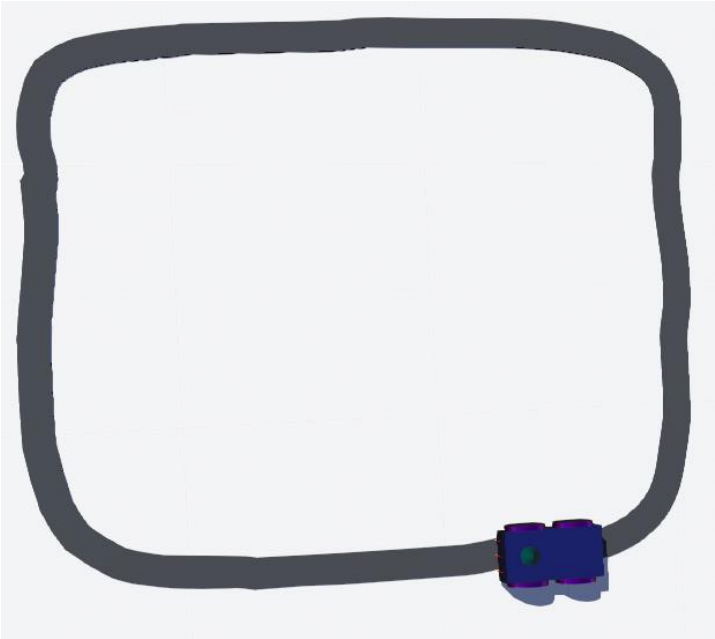

Figure 7. Rectangle track with rounded corners made using TinkerCad

SANIC was then tested on tracks with sharp, angled corners, starting with a square-shaped track, shown in Fig. 8. While it was unable to stay on track with a speed of 3, lowering its speed allowed it to have more time for the error to build up and stay on track throughout.

Finally, a triangular-shaped track was tested, shown in Fig. 9, to investigate the extent of SANIC's ability to take very sharp turns. While the robot was able to go around sometimes, it would often go straight through the turn without attempting to change direction. This highlights the robot's main limitation, taking sudden sharp turns. SANIC cannot go around these turns as the error in the controller doesn't grow fast enough to accommodate the sudden change in direction. The robot, therefore, moves straight through the turn.

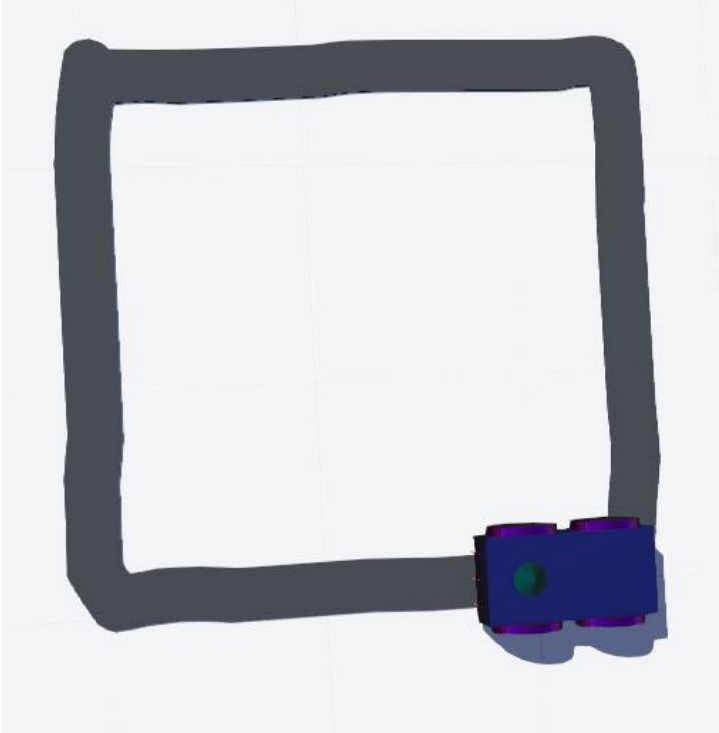

Figure 8. Square track made using TinkerCad

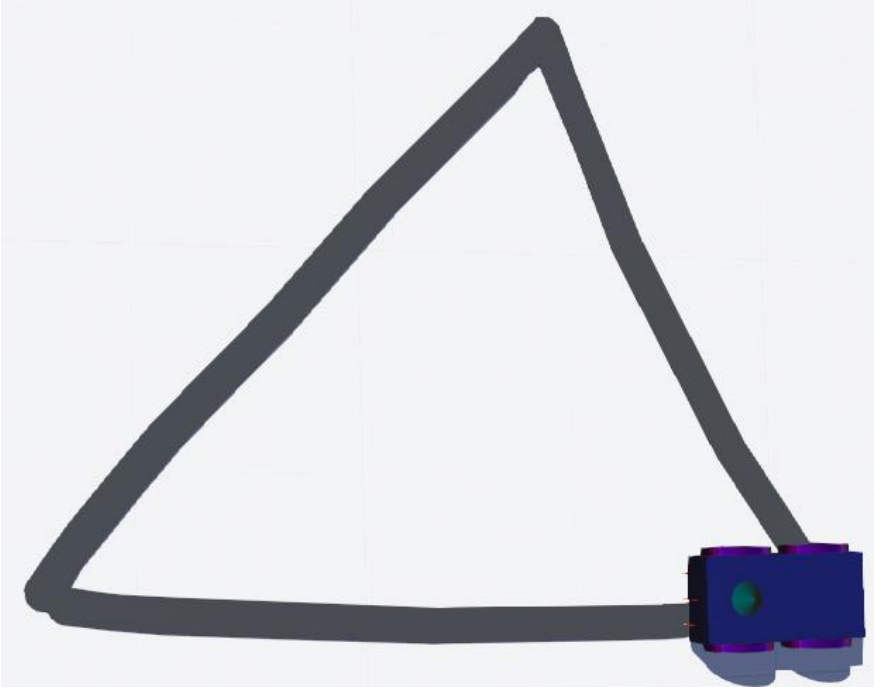

Figure 9. Triangle track made using TinkerCad

6. Conclusion

In this project, a robot and its controller were made from scratch and tested. First, the robot's body, wheels, and sensors were made of Webots base nodes and connected through hinge joints. Physics and bounding environments were added to each part to ensure proper functioning and interactions between SANIC and its environment.

A simple controller was then created as an initial attempt to control the robot. Using the three IR sensors, the robot's wheel speeds would change between their maximum velocity and 0 depending on the IR readings. While it functioned on an initial, peanut-shaped track, its motion was jerky and slow. In addition, it was unable to stay on track when it encountered sharp turns.

A simple PID controller was implemented to make the robot's movement smoother and allow it to move faster in different situations. Controller gains were determined using trial and error, and the robot was then tested on numerous tracks. Testing showed that SANIC could move around smooth, circular corners with ease. It often had issues with sharp, discontinuous changes in direction. Indeed, while it has minor issues going around F1 tracks, it was unable to go around a square track without reduced speed consistently and unable to follow a triangular track.

Nevertheless, this project was successful in creating a PID-controller robot. Initially, the robot was supposed to be a simple line-following robot with two sensors, using a pre-made robot model, going through, improving, and testing the project was highly enjoyable. Future improvements for this project could include adding a wall-following aspect to the robot's code and making it avoid obstacles by going around them.

7. Bibliography

[1] “Webots: robot simulator,” Cyberbotics Ltd. https://www.cyberbotics.com/

[2] “How Do IR Sensors Work?” Techwalla. https://www.techwalla.com/articles/how-do-ir-sensors-work

[3] Gada, K., “How to design a 2 wheel differential drive robot in Webots? // Webots tutorial 2 // Kajal Gada,” YouTube, November 2020. https://www.youtube.com/watch?v=ebGJzymXv-o

[4] “Webots documentation,” Cyberbotics Ltd. https://www.cyberbotics.com/doc/guide/using-python

[5] Gada, K., “Line Follower Webots Tutorial,” GitHub, J 2021. https://github.com/KajalGada/Youtube-Tutorial-Download-Material/tree/main/Line%20Follower%20Webots%20Tutorial

[6] Gada, K., “Webots Tutorial: Line Follower Robot using epuck // Controller code in Python,” YouTube, December 2020. https://www.youtube.com/watch?v=D0jhvFZJ5Ok

[7] Rafsan, T., “Line Following Robot Implementation Simulation [WEBOTS SIMULATION],” YouTube, September 2021. https://www.youtube.com/watch?v=ruIGhs8_sFo

[8] Crazzy AI., “Make line following robot(LFR) in 2 minutes using Webots,” YouTube, March 2021. https://www.youtube.com/watch?v=ZiRJDt-Wo8E

[9] Januka36., “Simulated Line Following Robot Using PID Controllers with Webots and Python,” GitHub, April 2021. https://github.com/januka36/Simulated-line-following-robot-using-PID-controllers-with-Webots-and-Python

[10] “AUTODESK Tinkecad,” AUTODESK. https://www.tinkercad.com