PID Controller for Mobile Robot Wall Following

Robert Barbulescu, MSc Intelligent Systems & Robotics, De Montfort University, Leicester

***Abstract* — This paper presents an implementation of a PID controller to control a robot within a structured environment in order to perform various tasks. Our paper describes how a PID controller can be implemented in python and how it can be used to improve accuracy during the implementation of the tasks. The Python API and CoppeliaSim were used to create our environment and control the sensors and motors of the robot in order to complete the tasks. All tests were performed in a simulated environment containing surrounding walls and several obstacles that were detected by a set of sensors and followed by the robot. Our robot was able to follow the predetermined path and avoid the obstacles in this environment.**

Keywords* — *Path Planning & Following, Obstacle Avoidance, Simulation, Mobile Robots, Pioneer 3DX, V-REP, Python.

I. Introduction

A (PID) or Proportional Integral Derivative controller is a common feedback loop method widely used in the field of robotics due to its ability to provide accuracy and tracking control, as well as stability, while having a simple structure and design. This was demonstrated by Y. Su and C. Zheng in [1], were they performed simulations on a two degrees-of-freedom (DOF) manipulator using a simple non-linear Proportional Integral Derivative (PID) controller to prove how this method can solve the finite-time regulation of uncertain robot manipulators. Furthermore, L. Pacheco and N. Luo outlined the performance of a control law based on PID for mobile robot local path-following in [2].

Designing and developing systems and algorithms that allow the robot to perform wall following is according to Ratnayake et al. "one of the important studies in mobile robotics" because those algorithms depend on the different environments which contain various types of walls and obstacles, furthermore, those algorithms and systems can have varying accuracy and reliability levels [3]. Moreover, wall following can be considered today still a primitive method of indoor navigation as it is highly limited. While mobile robot navigation systems can be integrated with (GPS) Global Positioning System in order to obtain the mobile robot real time position, this is mostly viable in an outdoor scenario, as it is not accurate for indoor navigation as shown in [4].

Another important concept is path finding, which is the process of planning the shortest route between two points and it is an important part of navigation for mobile robots. Depending on how much information is known about the environment we can classify path planning as *global path planning,* the process of deciding on the most suitable way in which the robot can be moved form a start location to an end location, or as described by Termizehic et al. "determining a path in configuration space between the initial configuration of the robot and a final configuration such that the robot does not collide with obstacles and the planned motion is consistent with the kinematic constraints of the vehicle" [5]. And *local or reactive navigation,* which does not require any prior information about the environment as the robot will react to the detected obstacle and changes its heading direction in real time to avoid any obstacle, such navigation is most suitable for use in dynamic environments [6].

An alternative to using GPS is *odometry*, which is relatively more precis for indoor navigation on short distances as observed in [7]. Odometry does provide higher accuracy rate in comparison to GPS, M. Taufiqqurohman and N. F. Sari's research shows an improvement of 0.05% in average error of data retrieval, with GPS having 0.18% and odometry 0.13% [7]. While that is an improvement, according to Haq et al. "odometry is strongly influenced by internal systemic factors and non-systemic factors which cause the accumulation of error", for better and accurate odometry, calibration of error is necessary [8]. In order to reduce systemic error, researchers in [8] implemented a PID control as a position corrector and varied the constants (*Kp, Ki, and Kd)* for the robot to reach its desired location. This approach is of particular interest for our work as we will be undertaking a similar method in order to perform our tasks.

II. Purpose

The work presented in this paper considers modifying an existing program we developed previously which is available on [9] by implementing a PID controller in order to achieve better and more accurate wall following. For the purpose of our work we created a program that allows the Pioneer 3DX mobile robot to wander randomly within our simulated environment until a wall is detected and follow it in a pre-determined direction. During our original work we created a randomized movement and used the robot's sensors with an implementation of various parameters to allow for the path following to be precise. Now, we will create a PID controller and integrate it with our program in order to allow for the wall following to be more accurate. Our initial results form the program implemented without the PID and the program in which we implemented a PID will be analysed to determine the success rate of our implementation. Finally, we will present alternative methods that can be used to achieve path following and obstacle avoidance with PID, as well as future work regarding our approach and how we can improve our results.

III. Environment

During our work we used a simulated environment created in CoppeliaSim [10], the map compromises a wall which is the skeleton of the path to follow, it contains lines that indicate the ideal path that the robot should follow, and a bounding box around the area so that the robot cannot escape. The map created can be seen below in Figure 1:

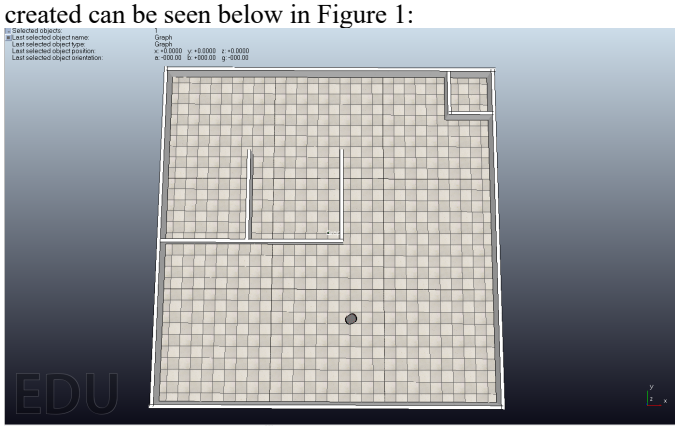


Figure 1 - Simulated Environment.

The surrounding box presented above compromises 15 by 15 meter walls, additional walls inside the box measure 8 by 4 meters, with the upper right-side corner box measuring 2 by 2 meters. All the walls and objects inside are static objects in order to ensure the robot is contained inside the map.

IV. System Description

A. Pioneer 3DX Mobile Robot

A fully programmable research mobile robot, widely used in mobile robotic researches for its versatility, reliability and durability, the robot can be used with a combination of various sensors such as: video camera, ultrasonic sensors, gyroscope, etc., and it is suitable for research and applications involving: mapping, teleoperation, localization, autonomous navigation, etc. The platform compromises motors with 500-tick encoders, 19cm wheels, an aluminium body, and it is also equipped with two sets of sonar sensors, 8 forward-facing ultrasonic sensors, 8 optional real-facing sonar sensors [5].

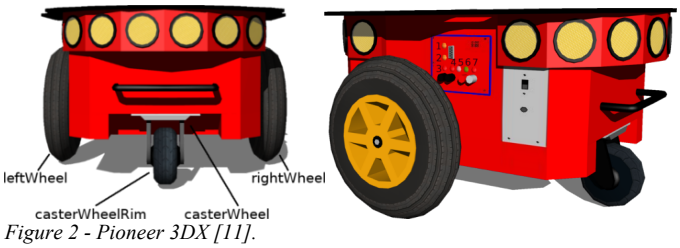


Figure 2 - Pioneer 3DX [11].

Different I/O ports are also available and can be used to connect up to 16 peripheral devices and power sources, there is an onboard computer with four RS-232 serial ports, Ethernet, PC104 bus with additional ports and a PSU controller, and all are accessible via an application connected to the operative system of the mobile robot [5].

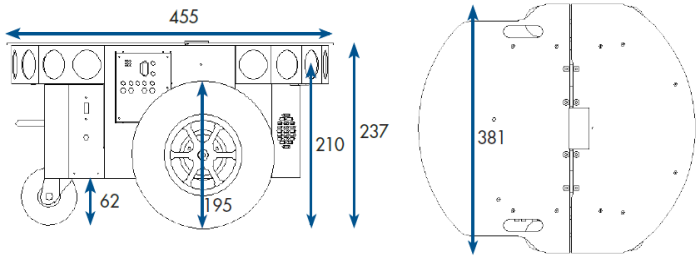


Figure 3 - Robot's Dimensions [12].

B. CoppeliaSim Edu (V-REP)

A robot simulator with integrated development environment based on a distributed control architecture, where each object/model can be individually controlled via an embedded script, a plugin, or a remote API client, making it versatile and ideal for multi-robot applications, with controllers that can be written in C/C++, Python, Java, Lua, Matlab or Octave [10]. For our work we will be using the remote API functions for Python to control our pioneer robot.

C. PID Controller

A PID controller is a common control loop feedback mechanism which calculates an error continuously (*e.g. the wrong speed of the motor spinning*) and applies a corrective action to resolve the error (*e.g. change power to the motor*) [13].

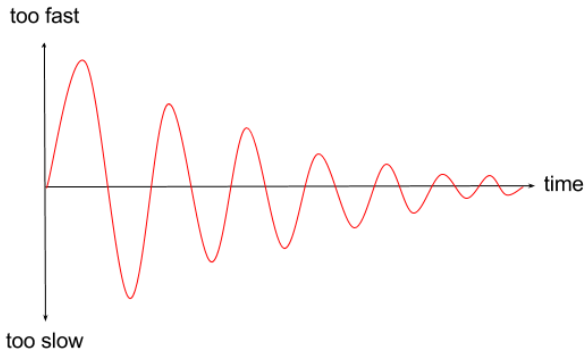


Figure 4 - Motor Speed Adjustment [13].

The PID controller will have an input value which will be the target motor speed needed to maintain, when encoder values are obtained, the value between the target speed and the actual speed will be calculated and if there is a difference, this is the error, it will apply an adjustment to the motor speed; Any adjustment overshoots at the next instance will be corrected by making a smaller opposite adjustment, this continues until the adjustments even out and the motors run at a constant speed [13].

The Proportional controller decreases steady-state error and enhances system accuracy, but decreases system stability, *the Derivate* controller improves settling and

system stability, and *the Integral* controller speeds up the process of the system approaching a set value and eliminate steady-state error [14].

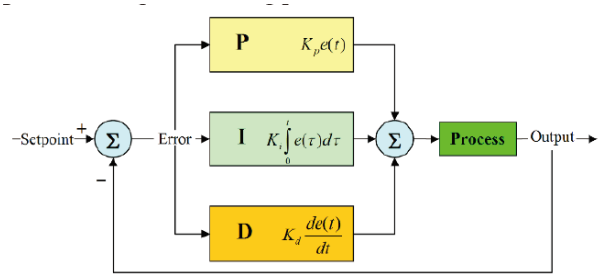


Figure 5 - PID Controller Structure [14].

From Figure 5 we can see the following components:

- P (proportional part) = $K_P \times e(t)$, where K_P is the gain of the proportional controller and $e(t)$ is the error at time t,
- I (integral part) = $K_I \times \int e(\tau) \times d(\tau)$, where K_I is the gain of the integral controller, e is the error and τ is the integration variable (time interval),
- D (derivative part) = $K_D \times de(t)/dt$, where K_D is the gain of the derivative controller, e is the error and t is the time.

V. System Design

A. Robot Controller

For the purpose of our task, the main objective of the robot is to maintain a fixed distance from the right-side wall and avoid obstacles in its path. We want to achieve this using the *Pioneer 3DX* mobile robot which is a differential drive robot, meaning, the movement is controlled by two separately driven motors.

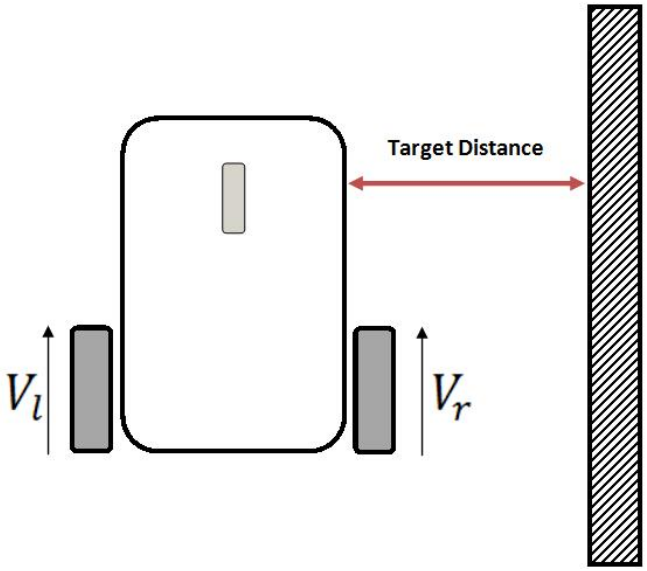


Figure 6 - Graphic Representation of the robot [16].

In the figure above, *Vl* and *Vr* represent the velocity of the right and left wheel which can be used in the following scenarios:

1. *Both wheels move at the same speed resulting the robot is moving in forward linear movement,*
2. *The wheels are moving at the same speed but one of them is moving in the opposite direction in order to achieve rotation at at the midpoint of the wheel axis,*
3. *If one wheel is moving and the other is set to a value of 0, then our robot will rotate, with center of rotation at the stationary wheel [16].*

We used several classes to define this in python and initiated the Pioneer's motors using the Remote Python API Function guide from [17].

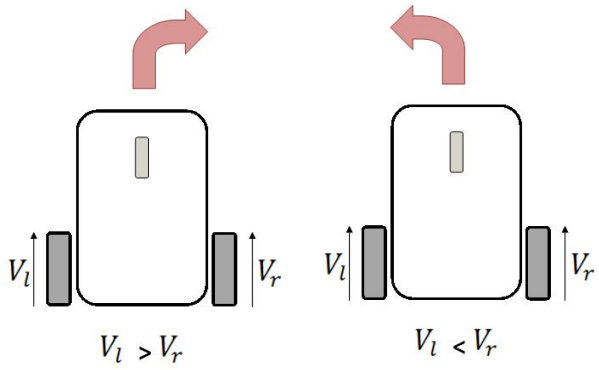


Figure 7 - Move right or move left depending on the speed of each wheel. [16].

Python motor initialization:

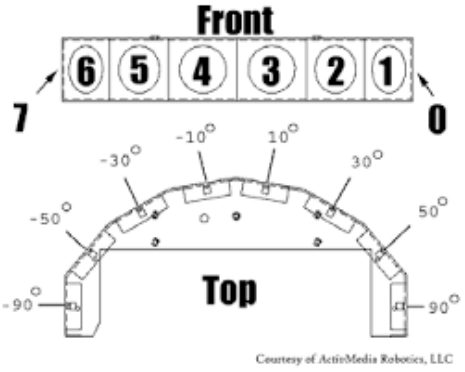
```
res, self.leftMotor = vrep.simxGetObjectHandle(clientID, 'Pioneer_p3dx_leftMotor',vrep.simx_opmode_blocking)

res, self.rightMotor = vrep.simxGetObjectHandle(clientID, 'Pioneer_p3dx_rightMotor',vrep.simx_opmode_blocking)
```

B. Obstacle Avoidance

Our design should be capable to follow the wall on the right side, for this we need to initiate the ultrasonic sensors and pass the readings through the PID controller.

For our program, in this instance, we activated sensors 0, 1, 6 and 15.



The robot will keep measuring the front and side distance from obstacles, if any detected the robot will turn left and calculate the error between target distance and measured distance. The error value will be passed to the PID controller which is used to control the speed of the left and right wheel.

C. PID Controller

As our starting point, we initialized all gains with 1 resulting in $K_p = 1$, $K_i = 1$ and $K_d = 1$, and continued by creating our variables needed for the PID controller:

```
current_error = 0
previous_error = 0
sum_error = 0
previous_error_derivative = 0
current_error_derivative = 0
```

Next, we created two more variables used to store the readings from the right side of the robot:

```
d1 = robot.getDistanceReading(robot.sideSonar)
d2 = robot.getDistanceReading(robot.rightSonar)
```

Since we are interested in following the right side of the wall, we needed to calculate the clearance between the robot and the wall, this was calculated using the following formula: *cos angle = d1/d2*, once we obtained the clearance we could calculated the current error.

```
dist = d1+d2/2
angle = d1 - d2
dist_to_wall = math.cos(angle) == d1/d2

current_error = dist_to_wall – dist
```

The angle is simply the distance between our side sonars and the distance to the wall is our target. This error was calculated to keep the robot in parallel with the wall. Using the calculated error, we implemented a PID controller to keep the robot parallel to the wall in the navigation path:

```
sum_error += sum_error + current_error*dt
current_error_derivative = (current_error - previous_error) /
dt
previous_error = current_error

cp += Kp * current_error + Ki * sum_error + Kd *
current_error_derivative
```

D. Wall Following

For each of the right-side sensors initiated previously, we implemented the following scenario:

- *If the sensor moves too close to the clearance distance calculated above the it will turn to the left to adjust using the values from the PID,*
- *If the sensor moves too far away from the clearance distance the robot will turn to the right to adjust using the values from the PID,*
- *If the sensor value is equal with the clearance, then it will forward with no correction at a pre-adjusted speed.*
- *If none of the above, then the robot will take a random turn.*

In order to obtain the ideal gains for our PID controller we used trial and error to test different parameters, furthermore in order for the robot to move forward we assigned negative values to the gains as our error was negative.

VI. Results

A. Expected Results

When using only the P controller, the robot should overshoot if the gain is too high, while for low gain value, there shouldn't be any noticeable effect on the robot. Our robot should successfully follow the wall with some amount of oscillation.

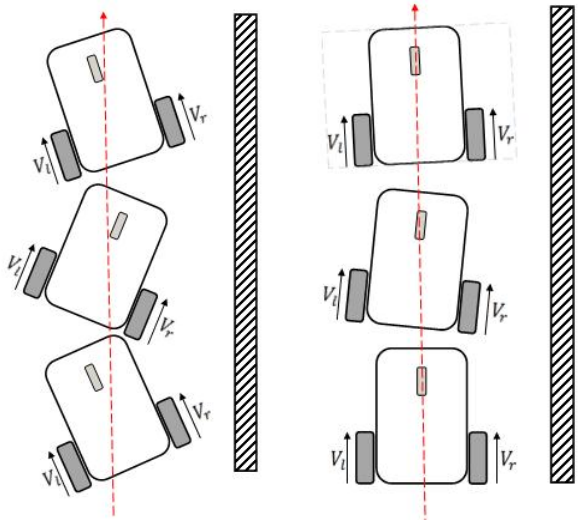


Figure 8 - Using P controller and PID controller [16].

When using the complete PID controller, the robot should adjust its movement and follow the wall accurately.

B. Actual Results

The robot using our PID controller follows the wall but overshoots at the corners, as well as it is unable to make 90 degrees turns. Furthermore, the oscillation is considerably high. In our original work, we used Finite State Machine (FSM) to achieve wall following, it responds to its input in different ways at certain times and is defined according to the behaviour or action that it is currently performing, the FSM can only be in one state at a time. Meaning, we used threshold values to limit the movement of the robot depending on the sensor reading.

For the purpose of this task, PID is meant to provide with smoother and more accurate wall following, our program did not achieve this and performed worst in contrast with our original work.

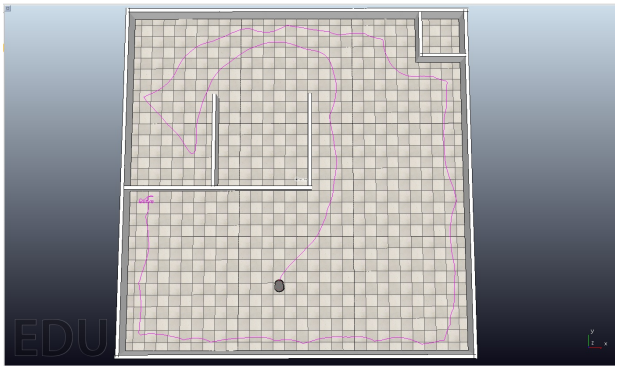


Figure 9 - PID Controller.

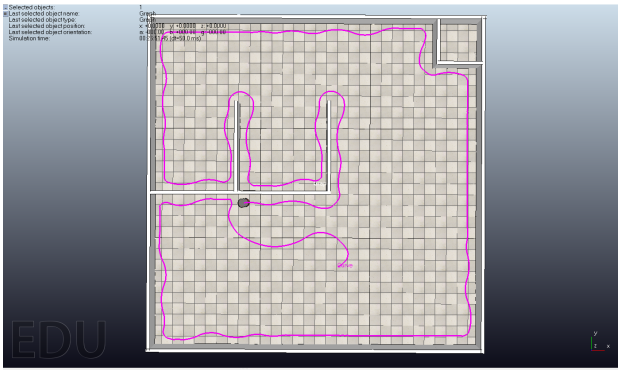


Figure 10 - FSM Controller.

Looking at **Figure 9** we can see the how the robot when other objects present, while is able to maintain the distance from the wall in scenarios that does not involve turning, other objects causes the sensor readings to overlap and changes the direction of the robot.

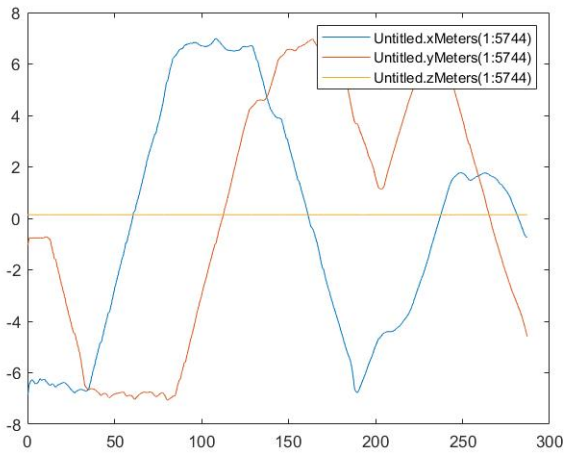


Figure 11 - PID Simulation plot.

Looking at the plotted data of the simulation for the PID Controller, we can see that the movement is less accurate than our FSM Controller, with more time spent overshooting corrections.

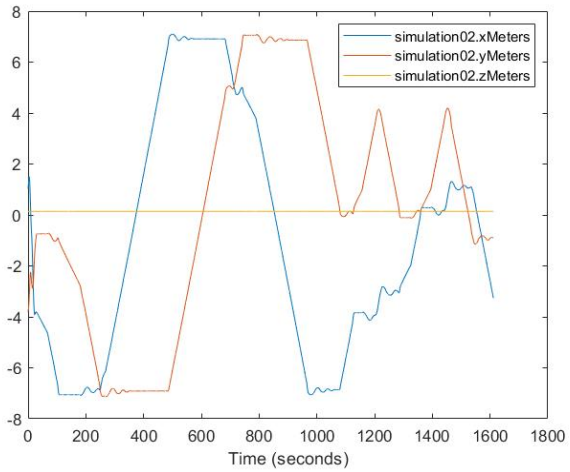


Figure 12 - FSM Simulation Plot.

VII. Conclusion

In this report we attempted at implementing a PID Controller in Python in order to follow the wall accurately, we used the Pioneer 3DX ultrasonic sensors to obtain data and passed to the PID controller for adjusting speed of wheels for accuracy. Several aspects have been discussed with an analysis of our expected results and actual results. While the robot achieved some degree of wall following it was inaccurate, and often overshot obstacles and corners due to sensor picking other objects and changing direction.

Our analysis is that the program currently lacks the ability to stay with the threshold clearance distance from the robot and the wall, this could be cause by several reasons, including our implementation of the wall following is flawed and requires improvements, our gains have not been adjusted correctly or the clearance distance calculation and method contains a logical error we did not considered. When compared with our original work, our current implementation does not bring any improvement and requires further work in order to address the problem mentioned above.

This implementation allowed us to use and research about the PID method for accurate adjustments, our program did not presented the desired results but it does provide a start for implementing a PID control in the context of wall following and obstacle avoidance.

VIII. References

[1]
Y. Su and C. Zheng, ‘A simple nonlinear PID control for finite-time regulation of robot manipulators’, in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 2569–2574, doi: 10.1109/ROBOT.2009.5152244.

[2]
L. Pacheco and N. Luo, ‘Testing PID and MPC Performance for Mobile Robot Local Path-Following’, *International*

Journal of Advanced Robotic Systems, vol. 12, no. 11, p. 155, Nov. 2015, doi: 10.5772/61312.
[3]
R. M. N. B. Ratnayake, T. S. de Silva, and C. J. Rodrigo, 'A Comparison of Fuzzy Logic Controller and PID Controller for Differential Drive Wall-Following Mobile Robot', in *2019 14th Conference on Industrial and Information Systems (ICIIS)*, Dec. 2019, pp. 523–528, doi: 10.1109/ICIIS47346.2019.9063333.
[4]
R. Siegwart and I. R. Nourbakhsh, *Introduction to autonomous mobile robots*. Cambridge, Mass: MIT Press, 2004.
[5]
T. Terzimehic, S. Silajdzic, V. Vajnberger, J. Velagic, and N. Osmic, 'Path finding simulator for mobile robot navigation', in *2011 XXIII International Symposium on Information, Communication and Automation Technologies*, Oct. 2011, pp. 1–6, doi: 10.1109/ICAT.2011.6102086.
[6]
Lim Chee Wang, Lim Ser Yong, and M. H. Ang, 'Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment', in *Proceedings of the IEEE Internatinal Symposium on Intelligent Control*, Oct. 2002, pp. 821–826, doi: 10.1109/ISIC.2002.1157868.
[7]
M. Taufiqqurohman and N. F. Sari, 'Odometry Method and Rotary Encoder for Wheeled Soccer Robot', *IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 407, p. 012103, Sep. 2018, doi: 10.1088/1757-899X/407/1/012103.
[8]
R. Haq, H. Prayitno, Dzulkiflih, I. Sucahyo, and E. Rahmawati, 'A Low Cost Mobile Robot Based on Proportional Integral Derivative (PID) Control System and Odometer for Education', *J. Phys.: Conf. Ser.*, vol. 997, p. 012046, Mar. 2018, doi: 10.1088/1742-6596/997/1/012046.
[9]
R. Barbulescu, *robertandreibarbulescu/IMAT5121-Mobile_Robotics*. 2020.
[10]
'Robot simulator CoppeliaSim: create, compose, simulate, any robot - Coppelia Robotics'. https://www.coppeliarobotics.com/ (accessed Nov. 25, 2020).
[11]
'Webots': https://cyberbotics.com/doc/guide/pioneer-3dx (accessed Nov. 25, 2020).
[12]
'Educational robots, service robots, programmable robots, front desk robot, robots for research', *Génération Robots*. https://www.generationrobots.com/en/ (accessed Nov. 25, 2020).
[13]
'Untitled'.
https://projects.raspberrypi.org/en/projects/robotPID/3
(accessed Nov. 26, 2020).
[14]
T. Wang and C. Chang, 'Hybrid Fuzzy PID Controller Design for a Mobile Robot', in *2018 IEEE International Conference on Applied System Invention (ICASI)*, Apr. 2018, pp. 650–653, doi: 10.1109/ICASI.2018.8394340.
[15]
M. Foukarakis, A. Leonidis, M. Antona, and C. Stephanidis, 'Combining Finite State Machine and Decision-Making Tools for Adaptable Robot Behavior', in *Universal Access in Human-Computer Interaction. Aging and Assistive Environments*, vol. 8515, C. Stephanidis and M. Antona, Eds. Cham: Springer International Publishing, 2014, pp. 625–635.
[16]
~ Zxlee, 'Wall Following Robot', *Zx Lee*, Jun. 21, 2014. https://iamzxlee.wordpress.com/2014/06/21/wall-following-robot/ (accessed Nov. 30, 2020).
[17]
'Remote API functions (Python)'. https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm (accessed Nov. 30, 2020).