Q-Learning for Adaptive PID Control of a Line Follower Mobile Robot

Technical Report · December 2017

CITATIONS
0

READS
460

1 author:

Canberk Suat Gurel
University of Maryland, College Park
10 PUBLICATIONS **0** CITATIONS

SEE PROFILE

UNIVERSITY OF MARYLAND

2017

Q-Learning for Adaptive PID Control of a Line Follower Mobile Robot

FINAL PROJECT REPORT

CANBERK SUAT GUREL

Abstract

In the field of mobile robotics, wheeled robots are widely used due to their advantages over their legged counterparts. PID controller is a model-free control strategy that is often implemented in the low-level control of the mobile robot control hierarchy. Although PID controllers are the most employed controllers in robotics, their performance is greatly dependent on how well the PID controller parameters are tuned. In the late 1970s and early 1980s, when the analog controllers were replaced by digital controllers, one of the advantages that came with the digital controllers was the auto-tuning function [1]. Existing auto-tuning technologies are based on the step response analysis and frequency response analysis [1]. These kind of classical controller-tuning techniques are not suitable when the system is not fully known and the operation conditions are variable [2]. The field of machine learning promises an alternative approach to auto-tuning of PID controllers, which allows the PID control parameters to be tuned even when the system is not fully known and operation conditions are variable. In this paper, a Q-learning algorithm for adaptive PID control of a line following mobile robot is presented. The robot is modeled in the Simulink environment and a set of experiments were undertaken on the modeled test track. The experiment results give an idea about the advantages of machine learning approach to optimize the control of robotic systems compared to systematic manual-tuning approach.

Table of Contents

1. Introduction

Wheeled mobile robots have a number of advantages compared to their legged counterparts such as structural simplicity, energy efficiency, high locomotion speed and low cost of manufacturing. One of the most widely used design is the two-wheel differential drive robot that is composed of two-actuated wheels and one passive caster wheel, which statistically balances the robot [3]. This particular design is advantageous in terms of maneuverability as it can rotate on the spot when equal and opposite angular speeds are applied to the two wheels [4].

The control hierarchy of wheeled mobile robots is often categorized as high-level and low-level. In high-level control, one of the three major control paradigms, (e.g. hierarchical, reactive, and hybrid) are applied to undertake a motion task such as point-to-point motion, path following, trajectory tracking, obstacle avoidance, and wall following. The hierarchical control architecture requires a complete world model to plan an action based on the sensor data. Due to its high computation requirement, the hierarchical control architecture is slower to response. The reactive (i.e. reflexive) control architecture does not have a planning stage; it executes an action based on the sensor data hence it is quicker to response.

In practice, even two identical motors have a different response to the same applied voltage, i.e. the differential drive robot does not go straight when the same PWM applied to motors. Evidently, if the motors do not operate as anticipated by the robot, any task that requires the navigation of the robot becomes pointless. The low-level controller is a closed loop feedback controller, which measures the angular velocities of the two motors and applies the necessary control law to match the operating regions of the two motors. As a result, the robot navigates with a finer accuracy.

Line following task is an example of reactive control, which can be implemented to a wheeled mobile robot using a PID controller where a number of line sensors detect the position of the robot with respect to line and the necessary control action is applied to wheels to keep the robot on the line. PID controllers do not require a model of the system, they are simple to implement and the computation speed is relatively fast. However, the performance of the PID controller is greatly dependent on the choice of controller parameters. There are a number of controller tuning rules proposed in the classic control theory literature, nevertheless, these techniques are not suitable when the system is not fully known and the operation conditions are variable.

Reinforcement learning is a powerful approach when it is applied to robotics as it enables the robot to discover an optimal behavior through trial-and-error interactions with the environment [5]. Adaptive PID controllers based on the reinforcement learning approach

have a great potential to resolve the problem of controller tuning for mobile robots operating in environments that are not fully known or in varying conditions [2].

This report is organized as follows, in Section 2, a discussion about previous related research is presented. In Section 3, the theory behind reinforcement learning is discussed then adaptive PID control approach that is developed to address the problem with tuning the controller parameters are presented. In Section 4, the implementation of Q-learning algorithm is demonstrated. In Section 5, the results obtained using manual PID tuning and the results obtained by tuning the controller parameter with the proposed Q-learning algorithm are presented. In Section 6, a discussion about why these results might have been obtained is discussed. In Section 7, a brief summary of the project and a few final remarks are given. The paper is concluded with Section 8 where a number of future work that can further improve the project are presented.

2. Related Work

In the work of [2], an incremental Q-learning strategy is proposed for online learning of the optimal PID controller parameters of wheeled mobile robots. In the learning process, a temporal memory was used, which remains invariant while a specialization process is carried out to seek for the learning spaces of states and actions to reduce the computational load and mitigate the curse of dimensionality of the algorithm. Their experiment results show that the proposed strategy is successfully implemented in a real world robotic system.

In the work of [6], a policy gradient reinforcement learning method is proposed to tune the parameters of three low-level PID controllers of an omni-directional mobile robot. Their computer based experiment results demonstrate that the robot learned to follow a given path. In the work of [7], a similar approach was implemented to maximize the locomotion speed of a legged mobile robot by finding a set of optimal leg motion parameters. Their experiment results show that the robot achieved the fastest gait after 3 hours of training.

In the work of [8], a Q-learning algorithm was used to tune the PID controller parameters of soccer robots then these robots were subjected to a set of experiments against robots that are tuned by using the Ziegler-Nicholas method. The experiment results demonstrated that the robots that are tuned using reinforcement learning with Q-learning method had a 1.5 times faster response time and better stability. The downside of this implementation is that the entire action domain and the finite state transitions needs to be provided to the robot as a priori information.

3. Approach

The reinforcement learning approach allows the robot to learn a control policy through the interactions between the agent and its environment. Figure 1 demonstrates the framework of reinforcement learning, where an agent takes a series of actions A_t, each of which generates a reward R_t and a new state S_{t+1} [9].

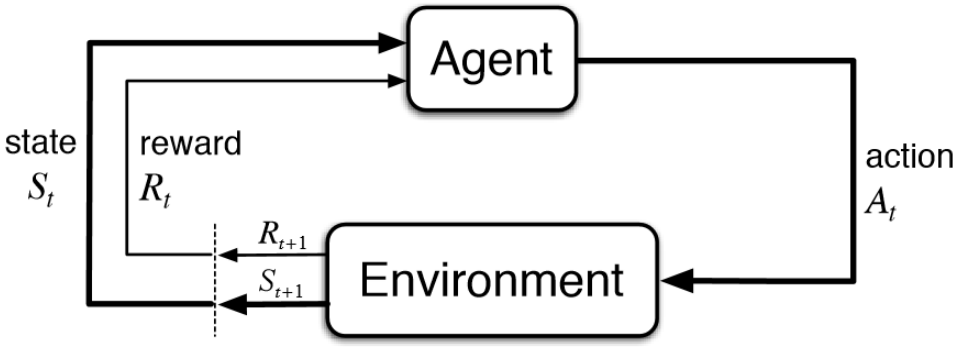

Figure 1: The reinforcement learning framework, taken from [9].

In 1989, Watkins presented Q-learning in his thesis [10] and provided a proof that it converges to the optimal value function. Q-learning is a form of temporal difference learning that is a model-free reinforcement learning method that combines elements of dynamic programming and Monte Carlo estimation [11]. Q-learning uses the state-action value

$$\mathrm{Q}(\mathbf{x}_t, \mathbf{k}_t) = \mathrm{Q}(\mathbf{x}_t, \mathbf{k}_t) + \alpha\left[r_{t+1} + \gamma \max_{\mathbf{k}} \mathrm{Q}(\mathbf{x}_{t+1}, \mathbf{k}) - \mathrm{Q}(\mathbf{x}_t, \mathbf{k}_t)\right]$$

to directly approximate the state-action function

$$\mathrm{Q}^*(\mathbf{x}_t, \mathbf{k}_t) = r_t + \gamma . E_{\mathbf{x}_{t+1}}[(V^*(\mathbf{x}_{t+1})|\mathbf{x}_t, \mathbf{k}_t)]$$

Then, the optimal policy can be obtained through

$$\pi^*(\mathbf{x}) = \arg\max_{\mathbf{k}} \mathrm{Q}^*(\mathbf{x}, \mathbf{k})$$

without the need for a model of system dynamics and state transition probability.

The mobile robot (i.e. the agent) interacts with the track by taking an action k_t, which in our case is a set of controller parameters that the PID controller uses to compute a control action u_t. The control action is then used to adjust the PWM signal applied to the motors and the system evolves from state x_t to state x_{t+1}. As a result, the agent receives a numerical signal, r_t so-called reward, which indicates how good/bad the action taken at state x_t was. The main objective of the agent is to learn the optimal policy π^*, which defines the set of optimal control parameters k_t for different states x_t that maximize the reward received over time.

$$k_t = \pi^*(x_t)$$

Figure 2 shows the adaptive PID control architecture that was used to find the optimal set of PID controller parameters for a given state and PWM value.

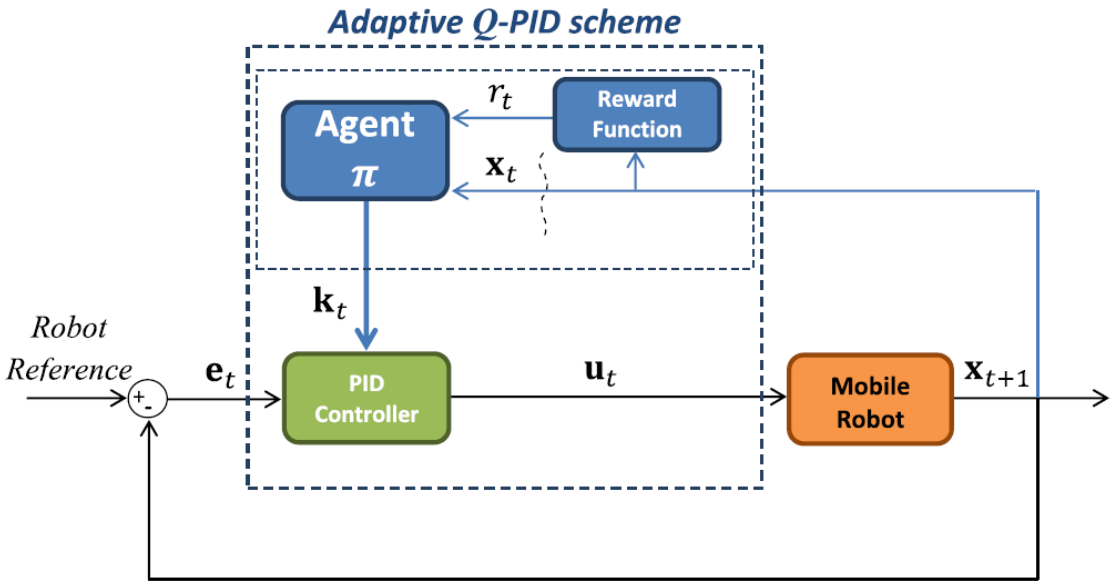

Figure 2: Adaptive PID control architecture, adapted from [2]

4. Implementation

A virtual mobile robot is modeled in the Simulink environment by deriving the forward kinematic equations describing the position and orientation of the robot in the XY plane in terms of the angular velocity of each motor. In order to make the model more realistic, a mathematic model of the motors are included and the motor parameters (e.g. internal resistance, torque constant, back EMF constant) can be adjusted in Code 1 (**Appendix**).

A virtual test track is created in Simulink to visualize the motion of the mobile robot during both the training stage and the testing stage. Figure 3 shows the virtual environment that is created. It can be seen that the test track consists of a ramp, a half circular turn (of radius 0.5m) and two quarter circular turns (of radius 0.3 m). When the robot is climbing the ramp, the resistant torque of the motors are increased and the robot slowed down to achieve more realistic results.

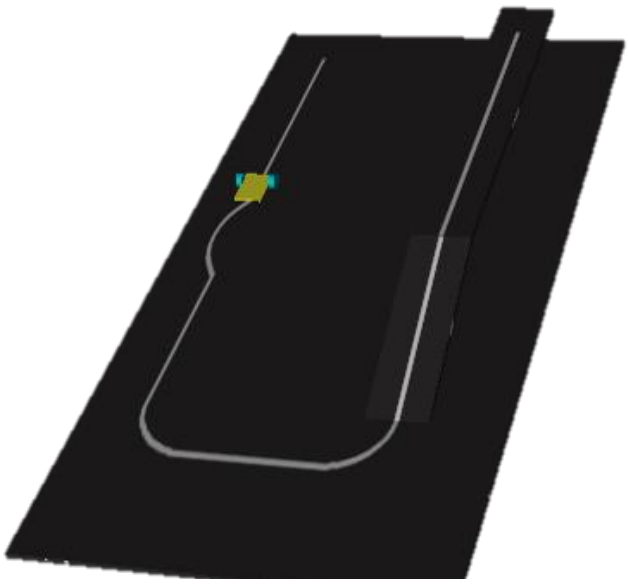

Figure 3: The virtual environment created in Simulink

In the model, the angular velocity of the two wheels are measured by encoders and then using the forward kinematic equations the position of the center of two motors and the direction of the robot are calculated. Since the relative position of the 4 line sensors that are located in front of the robot are known they are assigned with an analogue signal in the range of 10 to 510 depending on their closeness to the line. The sensors that output a signal greater and lower than 500 is considered HIGH and LOW, respectively. Then, depending on the HIGH and LOW sensors, the position of the robot with respect to the line is detected, which denotes the state x_t of the agent. The state x_t is in fact an error signal that tells the controller how far the robot is from the line. This error is the input to the PID controller, which is a feedback controller that calculates a control action u_t based on the chosen set of controller parameters k_t. Then, the control action u_t is summed with the PWM value of the right motor and subtracted from the PWM value of the left motor. The new PWM values are applied to the motors to minimize the measured error signal. Code 2 (**Appendix**) shows the PID controller implementation in MATLAB.

In the Q-learning algorithm, the state of the agent is determined depending on the size of error, and a reward value is assigned. Then, the maximum Q value in the row that is denoted by the given state is found as well as the column that correspond to the maximum Q value. The Q table is updated using the reward, a discount factor and the maximum Q value. Q-learning algorithm is given by Code 3 (**Appendix**). Initially the Q table was constructed in such a way that the columns of the matrix was representing every different combination of PID controller parameters. This approach turned out to be computationally inefficient due to the large memory requirement. In order to solve this issue a different approach was developed where the PID controller parameters are encoded inside the action k_t as a 7 digit number where different digits correspond to different controller parameters. The code that is used to encode this representation of controller parameters is given by Code 4 (**Appendix**).

5. Results

As discusses earlier, the model requires a predefined PWM value, which is altered by the PID controller to obtain the necessary change in angular velocity of the motors to keep the robot on line. For every different predefined PWM value, the PID controller parameters need to be retuned. A number of experiments with different predefined PWM values are undertaken and the fastest lab time for a given predefined PWM value is recorded. Initially, in order the test if the model works as anticipated, the PID controller is tuned manually when PWM was set to 300. The robot completed the track in 11.29 seconds. Figures 6 and 7 (**Appendix**) show the results of experiment 1. The manual tuning was a quite tedious process that took slightly under 4 hours. In the rest of the experiments, the Q-learning

algorithm trained the PID controller. Table 1 shows the fastest lab time achieved by the mobile robot for the given predefined PWM value.

	Predefined PWM	Training Time	Best Lap Time (s)
Experiment 2	500	1 h	6.7
Experiment 3	600	1 h	5.8
Experiment 4	600	1.5 h	5.77
Experiment 5	700	1.5 h	5.22

Table 1: Experiment result of Q-learning

Unlike the Tic-tac-toe assignment, in this case there is not a good indication of whether or not the optimal policy has been obtained. Therefore, the experiments are timed and the simulation is terminated at the end of allocated time slot. In experiment 4, the robot achieved a better timing when it is trained for 30 minutes more compared to experiment 3, indicating that the robot did not achieve an optimal policy in experiment 3. Figures 8-13 (**Appendix**) show the error and XY coordinates of the mobile robot in experiments 2-4.

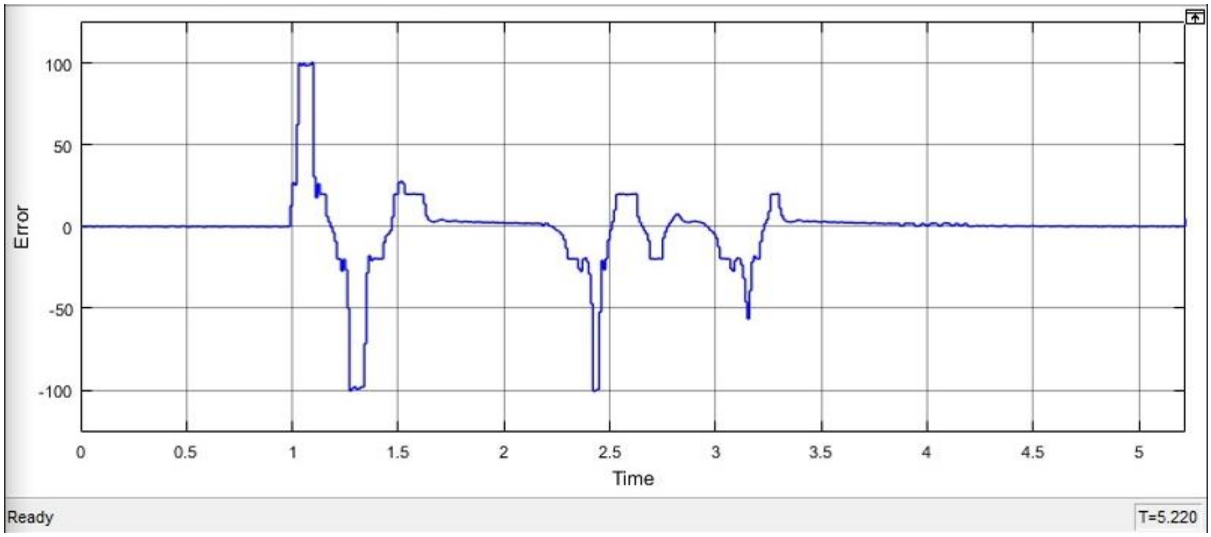

Figure 4: Error signal of experiment 5

Figure 4 shows the error signal of the fastest version of robot in experiment 5 when PWM was set to 700. It can be seen that the error signal peaked 4 times when the robot reached a turn in the test track. Figure 5 shows the XY coordinates of the robot in experiment 5. The red arrow shows an overshoot when the robot reached at the last corner. The experiments conducted with a predefined PWM value that is greater than 700 did not result with a successful run. The possible reason for that is explained in **Analysis** section.

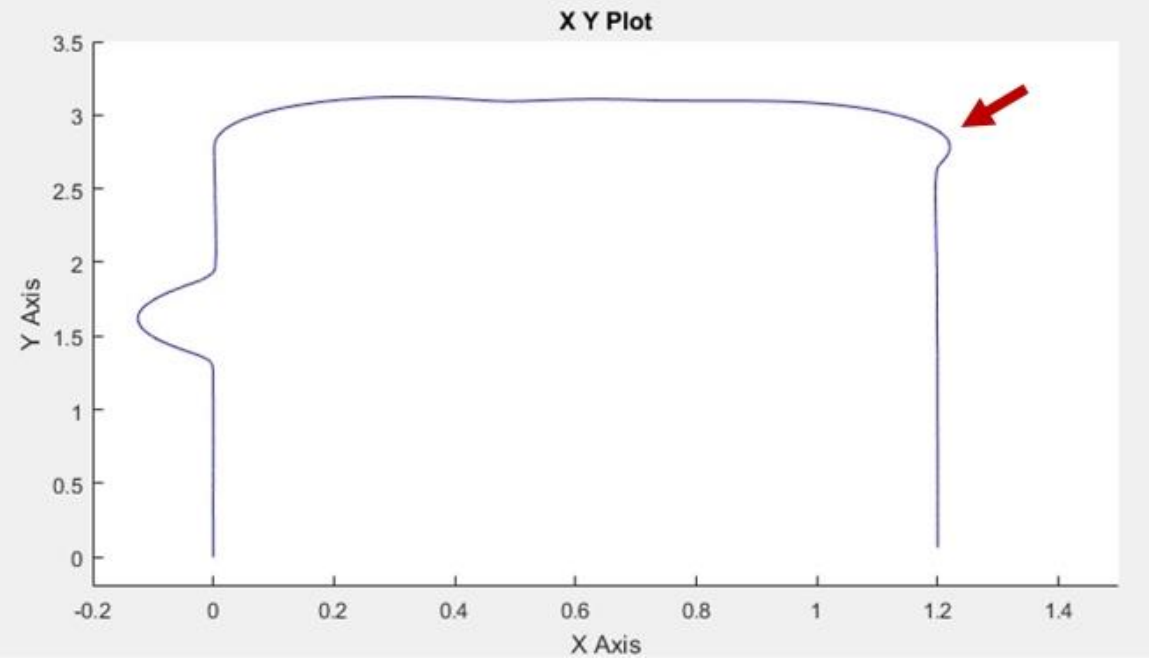

Figure 5: XY coordinates of experiment 5

6. Analysis

The experiments showed that the implementation of the Q-learning algorithm was successful as the Q-learning developed a robot that completed the track 2.16 times faster in a period that was 2.5 hours shorter compared to the manual tuning method.

In the experiments, a 10-bit motor driver board was used to apply the PWM signal to the motors hence any PWM value that is greater than $2^{10} = 1024$ is truncated to 1024. As it was stated in **Results** section, when the PWM value was set to 800 or greater, the controller output (control action is u_t forced to be between 0 and 224. However when the PWM was 700, the control action could take a value between 0 and 324 and hence it had a greater authority over the angular velocities of motors.

Evidently, when the predefined PWM value is increased the robot traverses faster in the straight line and therefore the controller must be more robust to supply a rapid change in angular velocity when the robot reaches a corner at the end of the straight line.

The experiments showed that when PWM was 800 or greater even if the controller selected the maximum value of the control action it was not enough of a change in angular velocity of the motors to keep the robot on the line. Consequently, all of the experiments resulted with the robot going out of bounce.

7. Conclusions

In this project, a model free reinforcement learning method was used to create an adaptive PID controller for a wheeled mobile robot to undertake the line following task. The proposed approach aimed to provide an alternative to manual and auto-tuning of PID parameters by making use of a machine learning algorithm, Q-learning, which is among the most commonly used and well-known reinforcement learning algorithms [11]. The experiment results proved that the Q-learning algorithm has successfully tuned the PID controller parameters of the mobile robot that was modeled in the Simulink environment. Hence, it can be said that the project has fulfilled the objectives that were proposed in the initial project proposal.

8. Future Work

Even though this project has successfully met its predefined objectives, it can still be improved by continuing to work on the implementation of machine learning algorithms to optimize the control of robotic systems. A few possible paths to be followed in the future are suggested as follows:

In the paper [2], Carlucho et al. presented a set of real world experiments by implementing an incremental Q-learning algorithm in a widely used differential drive mobile robot, Pioneer 3AT. In our project, a similar approach may be applied in future to engineer a physical mobile robot that self-tunes its PID controller parameters for the line following task.

Tuning PID parameters by Q-learning is just one of the methods of auto-tuning. Some of the other auto-tuning methods are empirical tuning (e.g. Ziegler–Nichols), model-based tuning, optimal tuning, and robust tuning. In the future, these methods may also be implemented to tune the PID controller parameters and the results may be compared against the tuning by Q-learning method.

As it was briefly mentioned earlier, due to the exponential increase in the requirement for computational power and memory, the action encoding method was introduced to encode the PID parameters in a 7 digit number. Although this method ameliorated the memory and computation power requirements it only allowed the controller parameters to be incremented by one. As a result, potentially a number of combinations of controller parameters that could have resulted with the robot reaching the end of the track are not tested. In the future, a more powerful PC may be used for achieving a finer step size or the Q-FA approach may be used to replace the Q-table method.

9. References

[1] T. Hägglund, "Autotuning," in *Encyclopedia of Systems and Control*, London, UK, Springer-Verlag London, 2015, pp. 50-55.

[2] I. Carlucho, M. D. Paula, S. A. Villar and G. G. Acosta, "Incremental Q -learning strategy for adaptive PID control of mobile robots," *Expert Systems With Applications,* vol. 80, pp. 183-199, 2017.

[3] B. Siciliano and O. Khatib, Handbook of Robotics, Springer , 2007.

[4] G. Oriolo, "Wheeled Robots," in *Encyclopedia of Systems and Control*, London, Springer-Verlag London, 2015, pp. 1548-1554.

[5] J. Kober, J. A. Bagnell and J. Peters, "Reinforcement Learning in Robotics: A Survey," *Internationational Journal of Research,* vol. 32, no. 11, pp. 1238-1278, 2013.

[6] A. Gloye, C. Goktekin, A. Egorova, O. Tenchio and R. Rojas, "Learning to Drive and Simulate Autonomous Mobile Robots," in *RoboCup 2004*, Heidelberg, Springer-Verlag Berlin, 2005, pp. 160-171.

[7] N. Kohl and P. Stone, "Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotaion," *Proceedings of the IEEE International Conference on Robotics and Automation,* pp. 2619-2624, May 2004.

[8] A. el Hakim, H. Hindersah and E. Rijanto, "Application of reinforcement learning on self-tuning PID controller for soccer robot multi-agent system," in *2013 Joint International Conference on Rural Information & Communication Technology and Electric-Vehicle Technology*, Bandung, Indonesia, 2013.

[9] R. Sutton and A. Barto, Reinforcement Learning: An Introduction, Cambridge, Massachusetts, London, England: The MIT Press, 2012.

[10] C. Watkins, "Learning from delayed rewards," King's College, Cambridge, UK, 1989.

[11] P. Stone, "Q-Learning," in *Encyclopedia of Machine Learning*, New York City, Springer New York, 2011, p. 819.

10. Appendix

Experiment Results

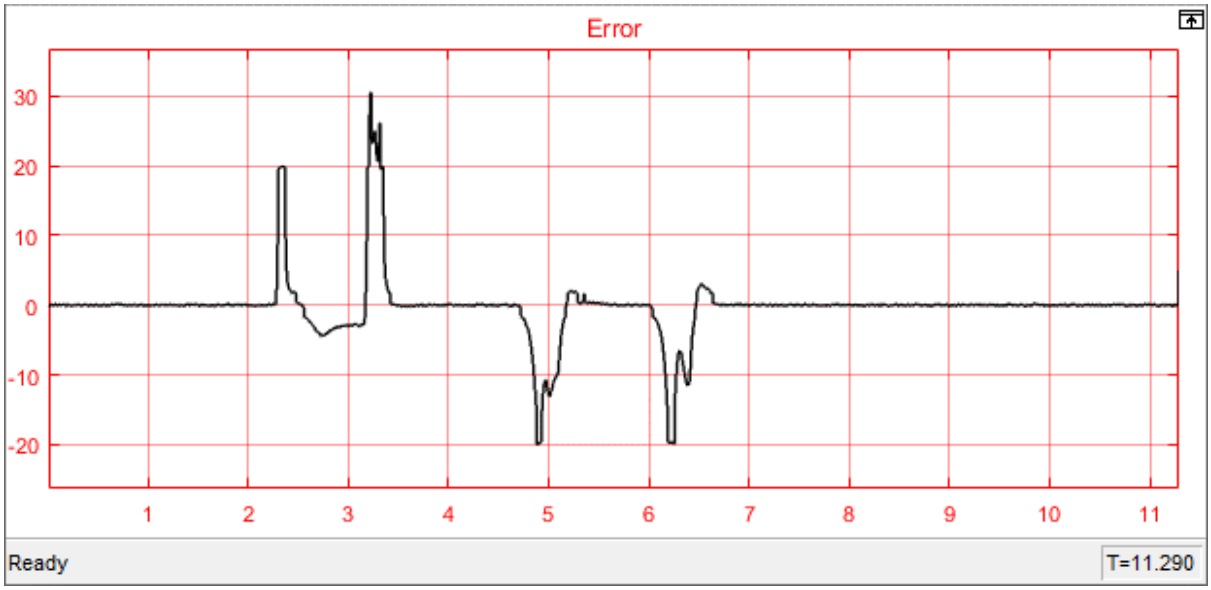

Figure 6: Error signal of experiment 1

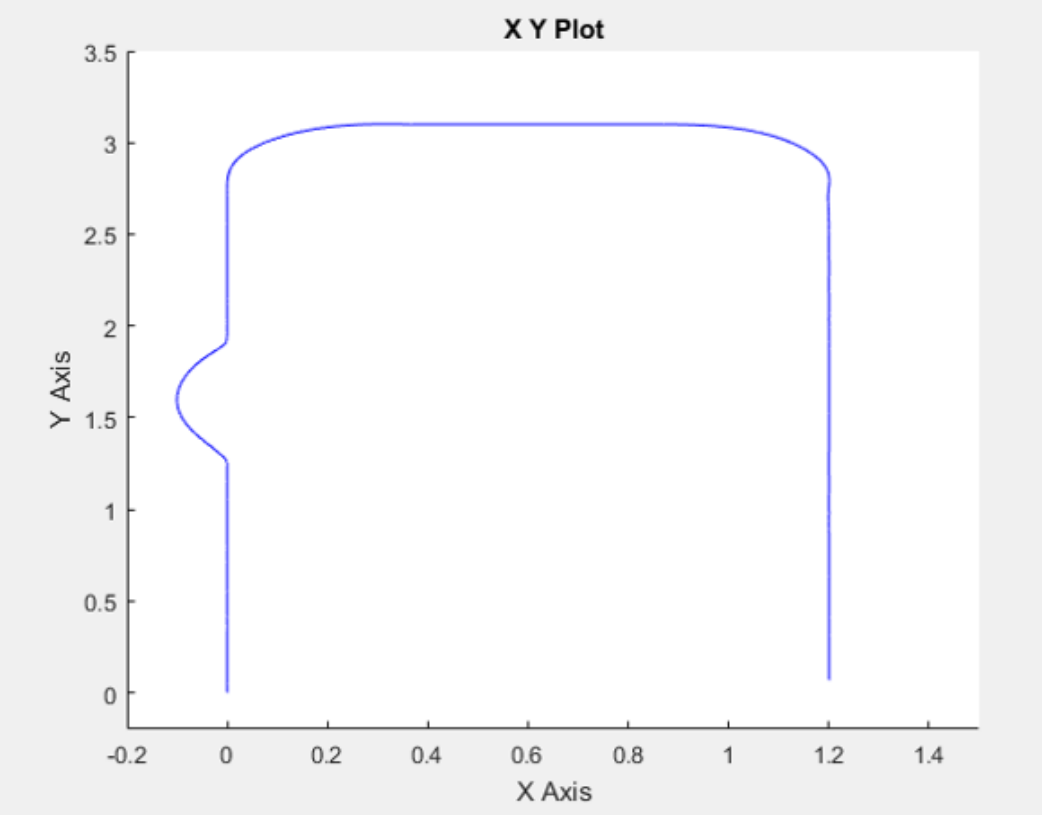

Figure 7: XY coordinates of experiment 1

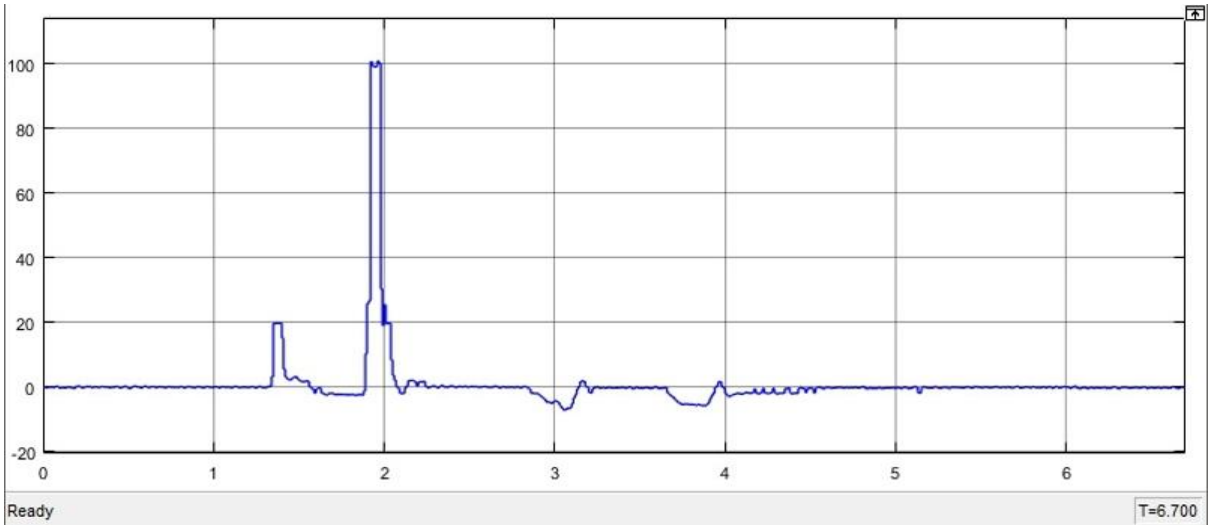

Figure 8: Error signal of experiment 2

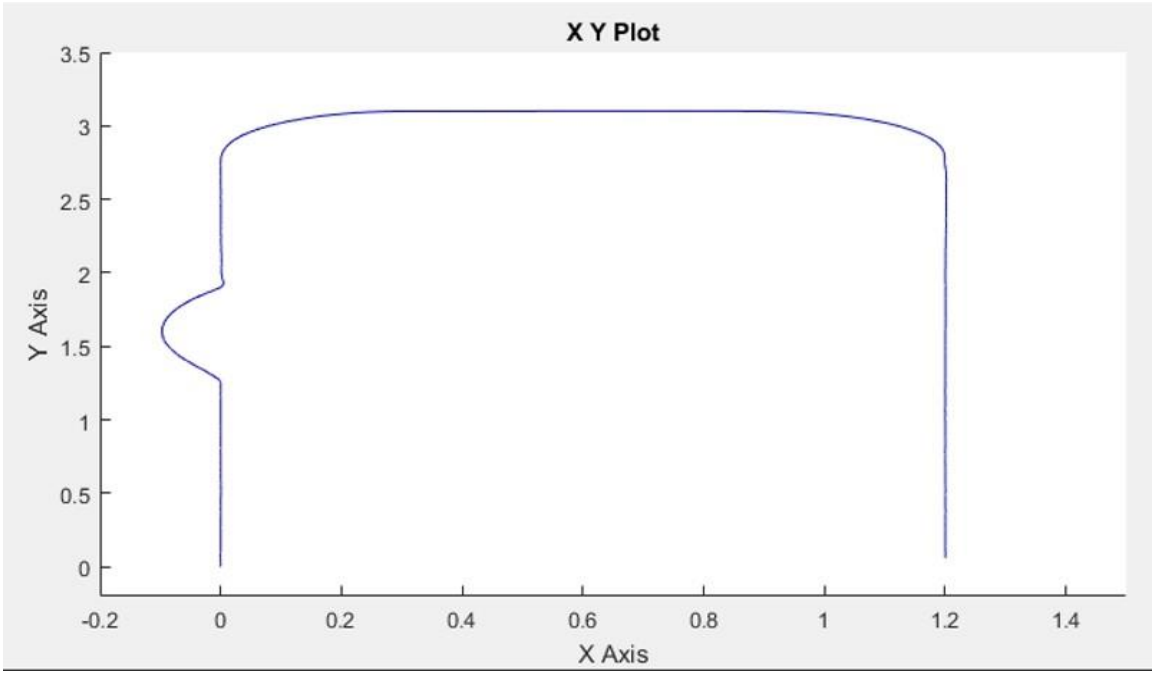

Figure 9: XY coordinates of experiment 2

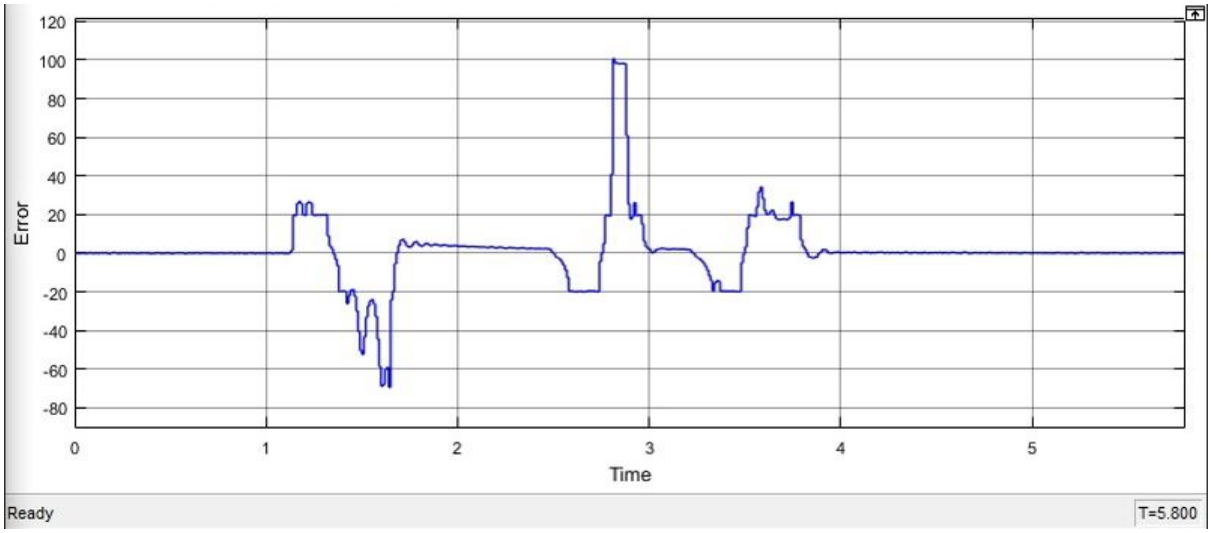

Figure 10: Error signal of experiment 3

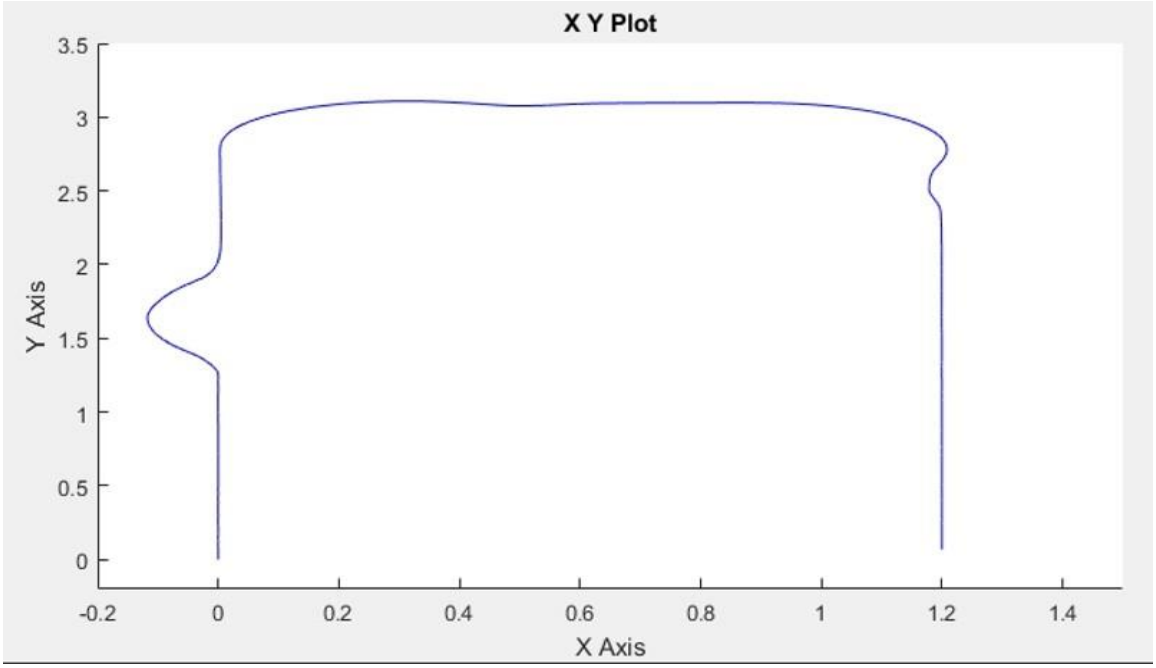

Figure 11: XY coordinates of experiment 3

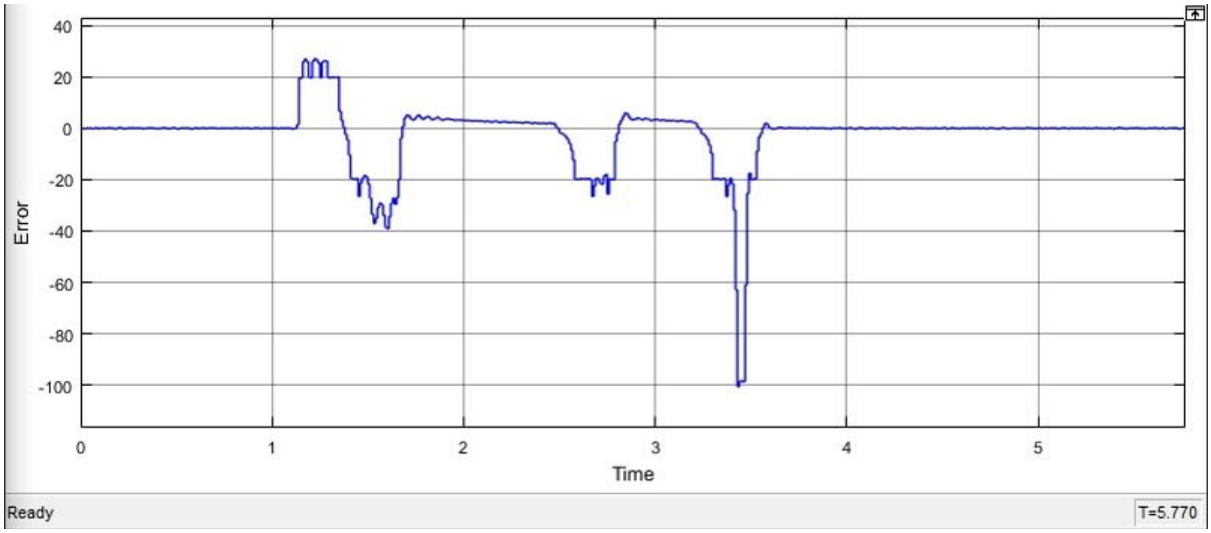

Figure 12: Error signal of experiment 4

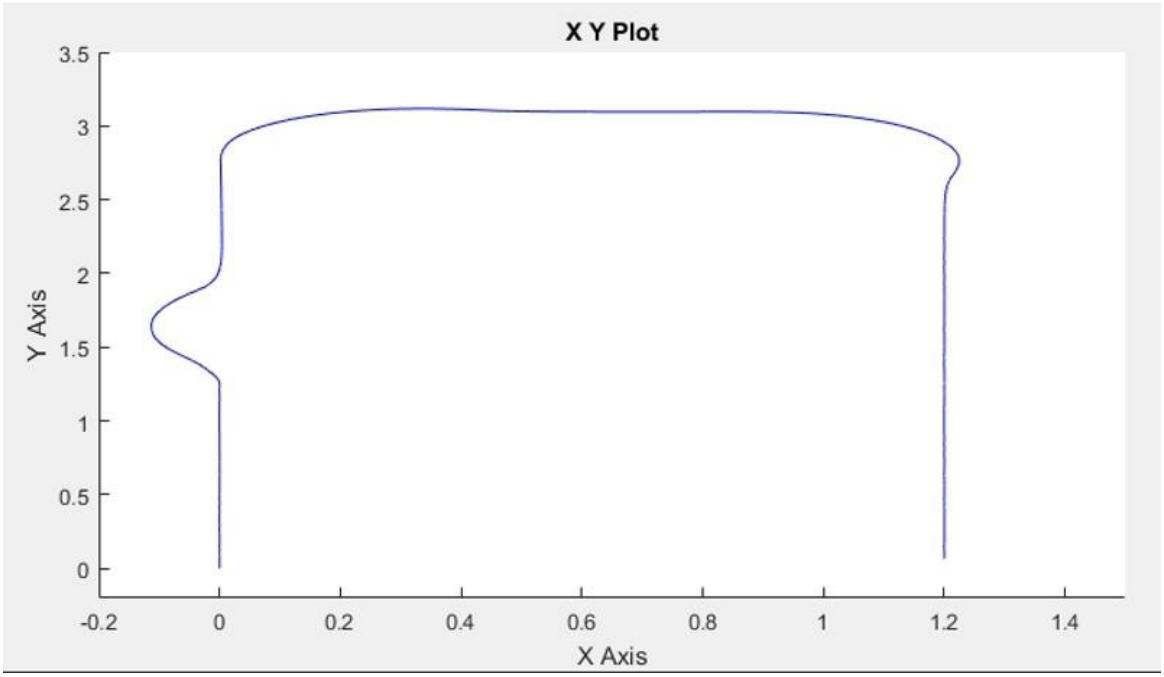

Figure 13: XY coordinates of experiment 4

Code 1: Initialization Parameters

```
% Q Learning Parameters
global Q
Q = zeros(7,100*100*1000);

previous_state = 0;
previous_reward = 0;

% Mobile Robot Hardware Parameters
wheel_distance=0.15;        % wheel distance (in m)
wheel_radius=0.03;          % wheel radius (in m)
u=0.1;                      % friction coefficient
m=0.5;                      % buggy mass (in kg)
G=30;                       % Gear ratio
ramp_angle=15;              % ramp angle (in deg)
Rm = 3;                     % motor resistance (ohms)
Kt=0.004;                   % motor torque constant (Nm/A)
Ke=0.004;                   % motor back EMF constant (Vs/rad)

% Software Parameters
% Drive Board Parameter
Bipolar=0;                  % drive board selection: '0' for Unipolar, '1'
for Bipolar
% Sensor Position Parameter
Dy1=+0.07;Dy2=0.07;Dy3=0.07;Dy4=0.07;           % sensor layout, y (in m)
Dx1=-0.035;Dx2=-0.015;Dx3=0.015;Dx4=0.035;      % sensor layout, x (in m)
ads1=1;ads2=1;ads3=1;ads4=1;                    % sensor output type
selection: '0' for digital, '1' for analogue
sample_frequency=100;                           % sensor sample frequency

% NOTE: the higher frequency is, the slower simulation will be.

% Simulation Parameters
J1=5e-7;                    % motor moment of inertia
J2=m*wheel_radius^2/G^2;    % buggy moment of inertia referred to motor
J=J1+J2/2;                  % total moment of inertia on one side
```

Code 2: PID Implementation

```
function [PWM_LEFT,PWM_RIGHT,stop,e_now]
=PID_control(S1,S2,S3,S4,e_pre,speed_left,speed_right, KP,KI,KD)

PWM=300;    %predefined PWM value

    if(S1<=11 && S2<=11 && S3<=11 && S4<=11)    %all sensors are LOW
        PWM_LEFT=0.0;   %stop the left motor
        PWM_RIGHT=0.0;  %stop the right motor
        e_now=5;
        stop=1;         %terminate the simulation

    else                                        %at least one sensor is HIGH
        e_now=((S1-10)+(S2-44)/5-(S3-44)/5-(S4-10))/5;      %assignment of
error based on the sensor readings
        propotional=e_now;
        derivitive=(e_now-e_pre);
        integral=e_pre+e_now;
        output=KP*propotional+KI*integral+(KD)*derivitive;  %control action
        PWM_LEFT=PWM-output;    %PWM of the Left motor
        PWM_RIGHT=PWM+output;   %PWM of the Right motor
        stop=0;
    end
end
```

Code 3: Q-learning Implementation

```
function [KP, KI, KD, state, reward] = Qlearning(e_now, previous_state,
previous_reward)
%#codegen
    global Q

    %e_now needs to be an integer for the following if statements
    e_now=int32(e_now);

    %depending on the size of the error the state and rewards are determined
    if ((e_now >= -1) && (e_now <= 1))
        state = 1;reward = 7;
    elseif ((e_now >= -5) && (e_now <= 5))
        state = 2;reward = 6;
    elseif ((e_now >= -10) && (e_now <= 10))
        state = 3;reward = 5;
    elseif ((e_now >= -15) && (e_now <= 15))
        state = 4;reward = 4;
    elseif ((e_now >= -20) && (e_now <= 20))
        state = 5;reward = 3;
    elseif ((e_now >= -25) && (e_now <= 25))
        state = 6;reward = 2;
    elseif ((e_now >= -30) && (e_now <= 30))
        state = 7;reward = 1;
    end

    %find the max row and column of Q for given state
    subMatrix = Q(state, :);
    [maxQ_value, max_column] = max(subMatrix);

    %Update Q matrix
    Q(state, previous_state) = previous_reward + 0.9 * maxQ_value;

    %find the set of actions corresponding to the max column
    [KP, KI, KD] = action_decode(max_column);
end
```

Code 4: Extraction of PID parameters from the Q value

```
function [KP, KI, KD] = action_decode(max_column)
      column=int32(max_column);    %max_column is a 7 digit number
      KP= rem(column, 100);        %last 2 digits is the KP parameter
      column = idivide(column, 100, 'floor');
      KI = rem(column, 100);       %4th and 5th digits is the KI parameter
      column = idivide(column, 100, 'floor');
      KD = column;                 %first 3 digits is the KD parameter
end
```