Reinforcement Learning Based Control and Navigation of Mobile Robots

Vishnu Hariharan Anand Naresh Hemadri Santhosh Kanaga Sabapathy Juvith Ghosh

***Abstract*—Waypoint navigation is one of the basic requirement of any autonomous agent. In this work we have implemented the end to end control problem of waypoint navigation task using Reinforcement Learning, replacing the traditional PID controllers. We have done rigorous experiments comparing the baseline PID method and the reinforcement learning based method. The experiments shows that the reinforcement learning based method performs well enough to be compared with traditional PID method. Also the experiments conclude that there is good scope for the reinforcement learning methods to outperforms the PID method in the future.**

I. Introduction

The basic functionality for any autonomous agent is to precisely navigate to its desired goal position from its current position (way-point navigation). To achieve this efficiently and reliably it is required to design controllers and path planning algorithms. PID controllers are most commonly used in robotics for navigation. But it usually take more than one PID in a nested way to do this task, which becomes a more complex and difficult task to tune all the PID gains.

Due to the success of reinforcement learning algorithms to solve complex control problems using DDPG [1], competing humans in Atari games using algorithms like Deep Q Learning [2], mastering board games using algorithms like alphazero [3] and muzero [4] shows that the reinforcement leaning algorithms are capable of solving a wide variety of tasks. This general learning approach to solve complex tasks encouraged us in implementing RL for waypoint navigation.

The reinforcement learning algorithms are broadly classified as Model based and Model free. In the prior method, the training of agent is done with a dynamic model of the environment. In our case it will be focused on model free RL algorithms where the model of the world is not present. Most of the RL algorithms falls in this category as the real world is unpredictable. Secondly, the RL algorithms can be classified as algorithms that work of continuous action spaces and algorithms that work on discrete action spaces. In our case we will be looking at continuous action space model free RL algorithms like DDPG [1], TD3 [5], PPO [6], TRPO [7] etc.

Deep Deterministic Policy Gradient (DDPG) is an model free algorithm that works of continuous action spaces and uses an Actor - Critic methodology where the architecture consists of an actor network which predicts the output and an critic network which evaluates the states. DDPG is used in our approach because it fits all our requirements and is easier to implement and test in a short period of time. Thus implementing DDPG helps us to evaluate fast whether RL algorithms could be used for waypoint navigation.

In our approach we have used 5 PID's acting in a nested way to efficiently achieve the waypoint navigation task in Romi mobile robot [8]. The 5 PID's tries to minimise the error in position (separate PID's of x and y coordinate frame), orientation, and the velocities of left and right motors. This approach is detailed in the Section II-A of this document. The RL agent aims to replace the positional and orientation PID leaving behind just the velocity PID.

A. Hypothesis Statement

Because of the inherent difficulty in tuning nested-PID controllers and due to the advancements of reinforcement learning algorithms in the field of control and navigation, we hypothesise that:

- RL algorithms could be an alternative to traditional PID controllers. In our experiment we aim to replace the position and orientation PID (figure 1) of the Romi using a neural network trained using RL.

We investigate this hypothesis through various structured experiments on the Romi mobile robot, comparing the performance of RL based controller with the baseline PID controllers for way-point navigation.

II. Implementation

In this section we will discuss in detail the implementation of both RL based and PID based methods for waypoint navigation using the Romi mobile robot.

The figure 1 shows the architecture of waypoint navigation of romi mobile robot using basic PID controllers. As shown, the PID based approach uses a set of 5 PIDs which is classified as Position PID, which controls x and y, orientation PID which controls the heading θ, and Velocity PID which controls the left and right motor velocities (v_r and v_l). The position and orientation PID combines to estimate the desired velocity for each motor. The RL Agent represent in the figure replaces the Position and Orientation PIDs. The encoders and kinematics shown in the figure estimates the instantaneous position, orientation and motor speeds of the Romi mobile robot.

Encoders: Romi mobile robot uses rotary encoders to know the count of each motor rotations. These counts can be converted to velocities which can be further used by forward

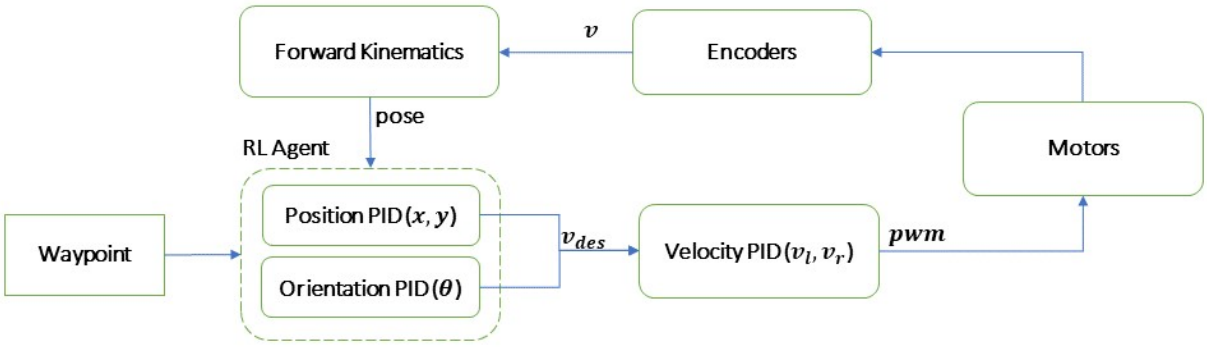

Fig. 1. Waypoint navigation

kinematics to estimate the pose. The conversion can be done using the following equation:

$$\begin{aligned} v_r &= \Delta n_r \frac{D\pi}{n'_r} \\ v_l &= \Delta n_l \frac{D\pi}{n'_l} \\ \mathbf{v} &= (v_r, v_l) \end{aligned} \tag{1}$$

where, v_r and v_l represents the left and right motor velocities in (m/s), D is the diameter of the wheel, Δn_r and Δn_l represents the change in count of left and right motor, n'_r and n'_l represents the count for one full rotation of wheel.

Forward Kinematics: The forward kinematics estimates the instantaneous position (x, y and θ) of the robot using the instantaneous velocity $\mathbf{v}$ of the motors, which is estimated by the encoders. The following equation is used for the pose estimation:

$$\mathbf{pose} = \begin{bmatrix} x_{old} \\ y_{old} \\ \theta_{old} \end{bmatrix} + \begin{bmatrix} cos(\theta) & 0 & 0 \\ 0 & sin(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{(v_r+v_l)}{2} \\ \frac{(v_r+v_l)}{2} \\ \frac{(v_r-v_l)}{L} \end{bmatrix} \delta t \tag{2}$$

where, L is wheel separation of the Romi, v_r and v_l are left and right motor velocity obtained from $\mathbf{v}$, and δt is the measurement time difference.

A. PID-Based

The PID controllers are non linear controllers used in various systems to exponentially reduce the error in measurement and demand of some controllable parameters of the system. The PID controllers have three main components, the Proportional term (P), Integral term (I) and the Derivative term (D). The proportional term tries to reduce the error in a linear way whereas the integral and derivative terms contributes to the non linearity of the controller. The general PID controller can be defined as:

$$PID = k_p e(t) + k_d \dot{e}(t) + k_i \int e(t) \tag{3}$$

where, k_p, k_i and k_d are the PID constants for P, I and D term respectively, $e(t)$ is the error i.e the difference between the desired and the measurement value. These need to be tuned manually to adjust the performance of the system. In this approach we have used 5 PIDs to control the error on x, y, θ, v_r and v_l.

$$\begin{aligned} v_x &= PID(e_x(t)) \\ v_y &= PID(e_y(t)) \\ v_\theta &= PID(e_\theta(t)) \\ v_{fwd} &= \sqrt{v_x^2 + x_y^2} \end{aligned} \tag{4}$$

where , e_x, e_y and e_θ represents the instantaneous error in x, y, and θ form current position of the Romi to the waypoint, v_{fwd} and v_θ represents the desired forward and orientation velocities in m/s required by the Romi to minimise the error, v_x and v_y represents the x and y component of the forward velocity. These velocities are combined to estimate the desired velocities for each motor.

$$\begin{aligned} v_{l_{des}} &= v_{fwd} - v_\theta \\ v_{r_{des}} &= v_{fwd} + v_\theta \\ pwm &= 255\,(PID(e_l(t)), PID(e_r(t))) \end{aligned} \tag{5}$$

where, e_l and e_r are the instantaneous error in motor velocities. The velocity PID is scaled by 255 to obtain the pwm values for each motors.

B. RL-Based

This method utilises an Reinforcement Learning algorithm to learn an optimal policy for performing waypoint navigation. The agent learns to reach the destination by maximising the cumulative reward. We have used Deep Deterministic Policy Gradient (DDPG) [1] algorithm for training. DDPG is an actor-critic, model-free RL algorithm that works on continuous action spaces. In our approach the RL agent learns to estimate the desired velocities of the left and right motor of the Romi, thereby replacing the position and orientation PID.

As the training requires a lot of time and also requires to reset the robot to the start position while training, it is practically not ideal to implement the training on the real robot due to various constraints like battery capacity, environmental

constraints and memory constraints on the robot to store experiences. Therefore, once the training is completed the actor network which estimates the desired velocity is transferred to the real hardware for testing.

Simulation

The entire simulation is done on Matlab. The figure 2 shows the simulation architecture used for training a RL agent for way-point navigation of Romi mobile robot. Two essential toolboxes are used in the simulation, one is the Reinforcement Learning Toolbox [9] which implements the DDPG algorithm and Mobile Robotics Simulation Toolbox [10] which runs the simulation of differential drive robots. The parameters of the Romi mobile robot such as the wheel radius and base length is given to the simulator to closely resemble the real hardware.

Waypoint: The way-points in figure 2 represents the destination point specified as x and y in the Cartesian coordinate. While training, the way-point is generated randomly during each resets within a radius of 1 meter thereby acting as the training datasets. Waypoints more that 1 meters are not considered because of the speed limitation of the romi to achieve the goal.

State: The state in figure 2 represents the current state of the robot which is defined by:

$$s_t = [e_x, e_y, \theta, \dot{e}_x, \dot{e}_y, \dot{\theta}, \mathbf{a_{t-1}}] \tag{6}$$

where, e_x, e_y, $\dot{e}_x$, $\dot{e}_y$ represents the error in x and y coordinate with respect to the way-point and its change from the previous step, θ, $\dot{\theta}$ represents the current orientation of the Romi and its change, and $\mathbf{a_{t-1}}$ is a vector of 2 elements which represents the action taken in the previous step.

The observation is chosen carefully so that the state is invariant of the starting position of the Romi because of the shift of coordinate system while calculating the errors. This makes the algorithm more generic which allows us to always reset the Romi to $(0, 0)$ always as its initial position while training.

Reward Function: The reward function is defined as

$$r = 3(d_0 - d) - 10d_\perp^2 \tag{7}$$

where, d is the distance from current position to way-point, d_0 is the initial distance at $t = 0$ and $d_\perp$ is the perpendicular distance of current position from the straight line i.e. the lateral deviation from straight line. The first term which measures the distance error outputs a positive reward if $d < d0$ i.e if the romi goes in the opposite direction its going to get a negative reward and the reward increases on reaching the waypoint and decreases if moving away form the waypoint. The first term is not squared because of this reason for getting negative reward. Whereas the second term look only at the deviation from the straight line. The deviation error will be zero if the current point is on the line and will get negative or positive error based on which side it got deviated. Since we are only concerned about the magnitude of deviation we have squared the term. The squaring will also help in maximising the reward in a faster way while training comparing when taking the absolute value because of the rate of change of increase or decrease of reward is higher while squaring when the error is more.

Network Architecture of RL Agent: The DDPG algorithm consists of an actor network and a critic network. The Actor network estimates the actions i.e. the desired motor velocities (in rad/sec) for the Romi to perform way-point navigation whereas the critic network estimates the value of being in that state, i.e. the Q-Function which obtained using bellman backup equation [1] given below.

$$Q^*(s, a) = \mathop{\mathbb{E}}_{s' \in \pi} \left[r(s, a) + \gamma \max_{a'} Q^*(s', a') \right] \tag{8}$$

where, $s' \in \pi$ represents the next state sampled under the policy π. The error in the estimated and true value given by eq 8 is backpropagated to correct the learnable weights of actor and citic network.

The figure 3 shows the network architecture for actor network. The Network consists of 3 fully connected layers with 16, 16, and 2 nodes respectively. We have used RELU [11] as the non-linear activation function for both the networks because it is computationally efficient, and does not have issues like vanishing gradient compared the other activation's like sigmoid etc. The hyperbolic tan function is used before the output layer because the range of the output need to in $[-1, 1]$. The input layer represents the state (s_t) of the romi and the output layers specifies the desired motor speed, thereby replacing the position and orientation PID.

Reset Conditions: The Romi resets itself to starting position regularly while training if either of the below mentioned criteria satisfies:

$$Reset = \begin{cases} d_\perp > 0.15 \\ d < -0.3 \\ T \geq 400 \end{cases} \tag{9}$$

where, T is the total time steps from the start.

Hardware Implementation

The figure 1 shows that the RL agent is replacing the position and orientation PID. This done by implementing the actor network in the microcontroller of the Romi.

Actor Network Implementation: Once the learning is done we can just transfer the actor network along with its learned parameters form simulation environment to the microcontroller of the Romi mobile robot. One of the key things to consider here are the memory capacity of the mircocontroller and the loop time it takes to run in the hardware. In this case we have used a minimalist network with just 3 fully connected layers with fewer number of nodes (16, 16, and 2). The replication of the network is done using the following equation:

$$\begin{aligned} h_1 &= relu(\mathbf{W_1} * s_t + \mathbf{b_1}) \\ h_2 &= relu(\mathbf{W_2} * h_1 + \mathbf{b_2}) \\ \mathbf{v} &= \tanh(\mathbf{W_3} * h_2 + \mathbf{b_3}) \end{aligned} \tag{10}$$

where, $*$ represents matrix multiplication, h_1, h_2 and $\mathbf{v}$ represents the hidden layers and output velocities respectively, $\mathbf{W_i}$ and $\mathbf{b_i}$ $\forall$ $i \in [1, 2, 3]$ represents the learned weight

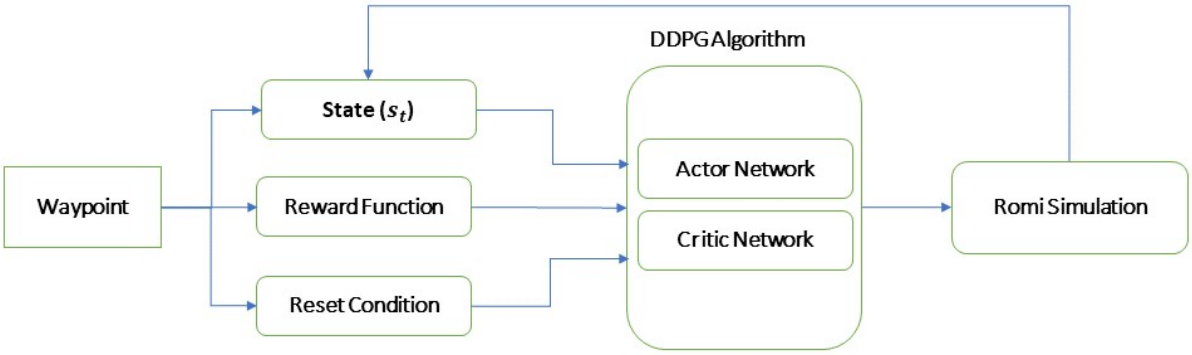

Fig. 2. Simulation architecture for training an RL agent.

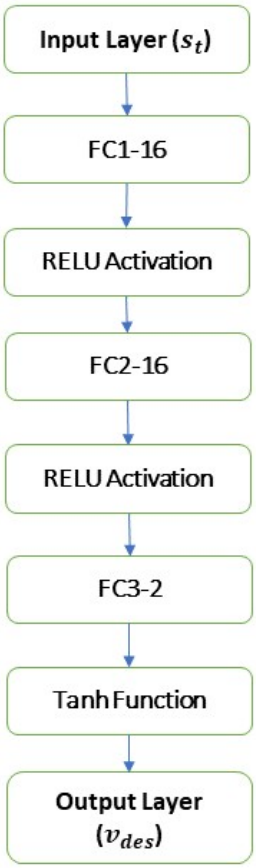

Fig. 3. Actor Network

matrices and biases of the 3 fully connected layers of the actor network.

III. Experiment Methodology

In this section we will be explaining in detail the methodology that we followed to conduct the experiments. Overall, there are three things that are considered: (i) Testing of Baseline PID method, (ii) Training and validation of RL based method and (iii) Implementation and testing the RL based method using Romi.

A. Overview of Method

Testing of PID Based algorithm

Once the baseline PID implemented on Romi as discussed in the section II-A, the only thing to do is to tune the PID values. Based on how well you tune the PID, it takes different paths to reach the way-point. In our results we have demonstrated this with 4 different variation of PID tuning just by adjusting the P-Term of the position PIDs.

Training and validation of RL Based Methods

Once the simulation environment is ready with all the components mentioned in Section II, the training of the RL agent can be started after configuring the hyper-parameters for tuning. Following hyper-parameters need to be tuned:

- Simulation test time: Tf
- Simulation sampling time: Ts
- Number of episodes for training
- Number of mini-batches per episode
- Training stopping criteria
- Learning rate for both networks

The total number of steps per mini-batch is defined by $T = Tf/Ts$ which is one of the criteria for reset conditions. Training stopping criteria is used to stop the training on attaining convergence, we usually use average cumulative reward above certain threshold as stopping criteria.

Once the training is done the results need to be validated in the simulation before testing on the hardware. In our experiments we will test the RL agent using multiple way-points both within scope of the training data provided and outside the scope in the simulator to validate the accuracy of the solution. In results a table is shown as a summary of these tests based on the success criteria of reaching the destination in comparison with the baseline PID implementation.

Testing of RL based method on Romi

Once the training is done, we can implement the way-point navigation algorithm on Romi as discussed in Hardware

Implementation of Section II. All the working way-point on the simulator are used as the test data sets in the robot. The path it traversed is recorded and compared with simulator and the PID based methods. The same experiment is conducted numerous times for same way-points and different way-points as well. The algorithm is also tested with multiple Romi's to test the robustness of the solution.

B. Discussion of Metric(s)

Multiple experiments is being done to evaluate the performance of way-point navigation. A set of 36 waypoints of at one meter radius from the origin and 10^0 apart were selected for conducting experiments. The waypoints are first test on the simulator and the once those are working are tested on 4 different Romis for collection of data. The data represents the path traversed by the romi recorded using the pose estimated in eq. 2.

Qualitatively the experiments are evaluated based on the graphs that depicts the path it took to reach the way-point. In order to quantitatively evaluate the way-point navigation, the metric should be capable of finding the deviation from the straight line (the shortest path) and also should evaluate whether it have reached the desired way-point.

Root Mean Square Error (RMSE) on the deviation from the straight line is used as one metric to compare the efficiency of the individual experiments. Each romi is tested all selected waypoints which are working and the mean of the RMSE is calculated for each romi. Again a mean and standard deviation is taken which represents the overall mean RMSE for all Romis and the average deviation from the mean. This overall mean and standard deviation is shown in the results.

$$RMSE = \mathbb{E}_{romi}\left[\mathbb{E}_{w \in W}|d_{\perp}|\right] \tag{11}$$

where, W is the set of all selected waypoints and $\mathbb{E}_{romi}$ specifies the expectation done on all the robots. The standard deviation is calculated using errors from each romi.

The advantages of RMSE on deviation is that it can be used to get the overall accuracy of the algorithms. The downside of this experiment is that it can only be used to measure if the experiment is a success (i.e. reaching the way-point). For example if in an experiment the Romi stayed at its initial position, the deviation is going to be zero at any instance, thus the RMSE on deviation doesn't make any sense. To get an overall comparison of how well the algorithms behaved we produced a table in the results that compares the success rate of the RL vs PID.

IV. Results

In this section we will be showing the results to evaluate the performance of the system as discussed in the section III-B.

The figure 4 compares multiple PID based approach with reinforcement learning based approach for way-point navigation. The figure depicts clearly that path taken by multiple PID approaches are different. This is obtained by changing the proportional gains of only x and y PIDs and all the other PID's constants unchanged. It can be also noticed that the best tuned system is PID-1 and it becomes worse chronologically. It can also be noticed that the RL algorithm can can be compared to PID-2 or PID-3.

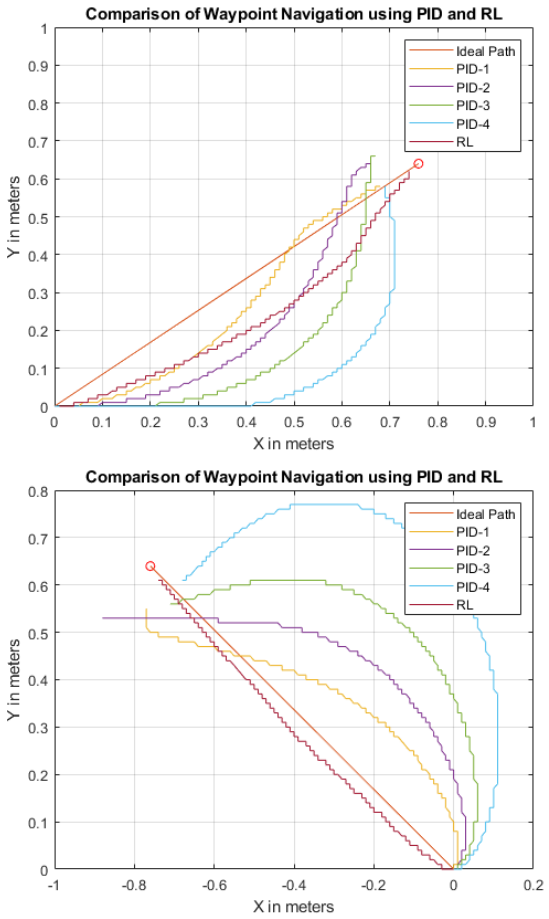

Fig. 4. This graph compares the multiple waypoint navigation approaches tested in Romi mobile robot.

Mertic	PID-1	PID-2	PID-3	PID-4	RL
RMSE	0.0557	0.0901	0.1372	0.1979	0.0802
Std-Dev	0.0068	0.0104	0.0172	0.0324	0.0136

TABLE I
THIS DATA DEPICTS THE COMBINED RMSE AND STANDARD DEVIATION WHEN TESTED AGAINST MULTIPLE ROMI.

The table I represents the combined overall measures for all the 4 Romis compared against 5 different waypoint navigation algorithms. The RMSE and standard deviation represents the overall mean and standard deviation of the RMSE of deviation from the selected waypoints. The data shows that the RL based method is somewhat similar to PID-2 algorithm.

The table II compares the success rate of RL and PID approaches. It is evident that the PID outperforms the RL approach by a huge margin. This data is based on testing done with 36 way-points. The PID is expected to perform well if the tuning is correct. The low success rate of RL based approach can be mainly due to the following reasons:

Algorithm	Success Rate (in %)
PID	100
RL	44.4

TABLE II
COMPARISON OF PID AND RL BASED ALGORITHM FOR REACHING THE WAYPOINT.

Fig. 5. The figure shows the progress while training on simulator

- The RL algorithm fails to improve after certain episodes. The red line in figure 5 depicts the average reward while training, it is getting almost saturated after 500 episodes of training. This can be due to lack to nodes in the neural network to further optimise the network.
- The reward function is a sparse representation of goal to be achieved. By adjusting the reward function much more precisely can improve the solution.
- DDPG is a basic algorithm that works on continuous action spaces. Some of its disadvantages is that it over estimates the Q function. Some of the advanced algorithm that works on continuous action spaces like TD3 [5], PPO [6] etc. can also be used to improve the results.

Romi's behaviour when using RL based method is little bit slow and also not smooth compared to PID based approaches. This could be easily improved using an updated reward function by incorporating some negative reward for sudden change in output actions. The speed also could be improved by providing a scaling to the output velocities while training. Also, the stopping condition that the RL agent learnt is to either to stop at the point or to rotate after reaching the waypoints. This is obvious from the reward function that the reward function has no component that checks the orientation.

The videos which showcases these observations can be found using the link: http://bit.ly/3qc7LDy.

V. DISCUSSION AND CONCLUSION

Because of the inherent difficulty of tuning nested-PID controllers and due to the advancements of reinforcement learning algorithms in the field of control and navigation, we hypothesised that:

- RL algorithms could be an alternative to traditional PID controllers.

The RL agent was able to replace the position and orientation PID of the Romi for some cases. The results shows that the Reinforcement Learning based controller requires a lot more improvement to improve its success rate of reaching the way-point. For the working cases it shows a comparable results with one of the best tuned PID. This contradicting results says that there is a potential for RL based approaches to perform better in the future.

In future, we would like to implement the same approach with higher number of hidden layers neurons using some on-board computer which has higher memory capacity than the microcontroller used in Romi. Also, we would like to implement way-point navigation with the other state-of-the-art reinforcement learning algorithms to compare results with this approach.

REFERENCES

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[4] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[5] S. Fujimoto, H. Van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," *arXiv preprint arXiv:1802.09477*, 2018.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[7] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, 2015.

[8] "Pololu corporation: Romi mobile robot. (https://www.pololu.com/product/4022)."

[9] "Mathworks: Reinforcement learning toolbox."

[10] "Mathworks student competitions team (2021). mobile robotics simulation toolbox (https://github.com/mathworks-robotics/mobile-robotics-simulation-toolbox), github. retrieved january 12, 2021."

[11] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.