Robo-Capstone Project Report
Control Policy using Deep Reinforcement Learning

Author: Manan Patel
Advisor: Dr. Fan Zhang

Abstract

With ubiquity in computational resources, Deep Reinforcement learning is now being used to model actuator control in robotics. However, it still suffers from the problem of sim-to-real gap and not acting with knowledge in rapidly changing environments. In this project, we try to develop a test bed for implementing Deep Reinforcement Learning and investigating the challenge of adapting the neural network policy to change in physics. The task of wall following robot was chosen as a control problem. We first try to learn a policy for wall following based on LiDAR input in simulated environment. Then, while deploying the system, we incorporate active learning by updating the weights of the nominal policy. With this approach we expect to validate if performing live update is at least not worse than, if not better, than a static policy.

1. Introduction

First introduced by Minorsky in 1922, PID control [3] is one of the common classical control techniques. It involves generating a control input which is proportional to the actuating signal, its derivative and its integral. Due to its simple nature, it is widespread in controlled industry settings where there is enough information about the environment. Moreover, trial and error techniques provide good results for a majority of such cases. Since, then several other techniques like pole placement and phase margin methods have also been introduced to mathematically calculate the gains, making PID control the most commonly used technique in the industry [4]. However, it heavily relies on the fact that it has near perfect plant model. The techniques mentioned above would fail if the model changes in real time or if there are significant disturbances to the system. Also,it assumes that the plant model is linear in nature. However, real world systems are non-linear in nature [2].

This provided motivation for non-linear control strategies since it improves upon linear control systems, helps in analysing hard non-linearities and takes into account model uncertainties with simplicity [9]. One of the methods of non-linear control is to first come up with the non-linear equations governing equations and then linearizing them around a typical working point. However, this method fails when the the non-linearities are significant. A more general method in this class is Lyapunov-based design in which, a stable system is designed by choosing a Lyapunov function V(s), where s is the state vector, and then selecting a state-feedback control law that renders the derivative of V(s) with respect to the states negative [8].

The choice between a linear and classical control can be a tough one as linear control has been studied extensively and many methods for analyzing it exist. On the other hand, albeit being more robust, non-linear control strategies involve complex mathematical analysis which can be cumbersome [9]. Furthermore, when we consider robots which usually operate in complex environments and have multiple degrees of freedom, applying the above approaches requires extensive initial cost of analysing the system.

Reinforcement learning on the other hand applies a completely different approach. Simply put, it contains an agent which interacts with the environment to attain a certain objective or a goal by performing actions. As a result of those actions, it accumulates rewards and/or penalties. Since the agent does not know what actions are ”good” to start with, it needs to explore the environment. However, once the agent has thoroughly explored the environment, the likelihood of finding a better state decreases. This is where reinforcement learning faces a primary challenge of the trade-off between exploration and exploitation which other methods do not encounter.

When the agent has full-access to the environment state, value iteration and policy iteration methods based on Bellman equation can come up with the optimal solution as described in [5]. Whereas in the case when the access to the environment is limited, temporal difference learning strategy is used as described in [1].

Deep Reinforcement learning, incorporates the above ideas with gradient descent of neural networks. The value function or policy function is estimated using an artificial neural network with non-linearities. The RL signal is back-propagated by taking the log probabilities of taking that ac-

tion. Research has shown that this setup is very powerful in coming up with solutions to RL problems with high dimensional input and output space. However, Deep Reinforcement Learning comes with the typical drawbacks of using an artificial neural network. Because of the black-box model, it is almost impossible to reason why a policy is behaving the way it is. Also, the network will not generalise well if the sample space is not sufficient enough and thus may fail suddenly if found in a state not seen by the network.

In this project, we will investigate deep reinforcement learning as it is ideal in case of robots. Developing control algorithm for robots might require significant initial cost in terms of research and understanding the physics of the robot. However, the black box model of deep learning is favourable here because it abstracts the dynamics away from the user.

2. Previous Work

Deep reinforcement learning in continuous action spaces is considered in the work in [10]. Exploration is crucial in RL, since it is the only way to discover new and better policies. The work in [10] discusses two methods for exploration in continuous spaces. The first method of exploration is called ϵ-greedy exploration which picks a random action with probability ϵ and the greedy current approximation for the optimal action is selected with probability 1 - ϵ. The second method is Gaussian exploration wherein the action that is performed is sampled from a Gaussian distribution with the mean at the action output of the algorithm. This method is shown to have better performance in their work. One of the reason why we adopt this approach is that while generating control output, "good" actions are usually closer to the most optimal action and thus, it would be advantageous to sample those actions first instead of randomly sampling over all of the action space.

A more recent work in the area of adaptive deep reinforcement learning is done by the authors in [12]. The game of curling is chosen as the control task for the robot which is a good test bed for studying the interaction between artificial intelligence machines and real world systems. This is because the game of curling is played on an ice sheet covered with small pebbles which change their conditions depending on factors such as the temperature, humidity, maintenance skill of ice makers, elapsed time since the conclusion of the maintenance, the previous stone trajectories, and finally the amount of sweeping done during the game—all of which neither are under control nor can be measured. These conditions imply that when delivering curling stones using exactly the same direction, force, and curl, the trajectory of the stones will inevitably vary over time. Their strategy involves using a set of adaptive features in a neural network and perform calibration throws to get an estimate of error between predicted and actual trajectory. These throws are then used to update the neural network which will then be able to account for the most recent environmental conditions in the play area. However, for our use case, we are following a model free approach and thus, we cannot estimate the trajectory of the robot to "calibrate" it. And thus, although the idea of this research is very close to the one considered in this work, it would be more applicable to Model Predictive Control.

Another such work is done in [7]. The authors mention that one way to perform online reinforcement learning is to train DNNs online by directly applying Backpropagation. However, this simple approach is expected to fall short. One key drawback of this approach is that it restricts the depth of the neural network. If the network is too shallow, it will not be able to model all the scenarios. Whereas if the networks is complex, it might not converge. In their work, they present a novel framework in the form of Hedge-backpropagation which involves adding "Hedge" weights at each layer which estimate the output given the input. The final output is then the a cumulative vote given by all the layers. Although this method is quite intuitive, it adds additional complexity while deploying it online. Also, their work considers neural network with 20 layers and does not mention if it is worthwhile for lesser number of layers. Since in our work, we are considering the simple task of wall-following, we will be working with smaller networks. Also, to have adaptability while maintaining key features, we can use the idea of transfer learning, that is, to freeze all layers except the last one and save computational costs.

3. Technical Approach

Reinforcement learning primarily involves the following: a policy, reward function, value function and optionally, a model.

A policy defines what actions the agent can take given a certain state. For this, the agent has to have some sense of the environment if not complete.

A reward function provides a mapping between state and its importance with respect to the objective that the agent wants to achieve. Most of the time, it is a function of the state the agent is in, but it can also be a function of state and action taken. The agents sole objective is to maximize the reward it gets in the long run.

A value function defines what is the maximum reward that the agent achieves in the future starting from that state. It is the predicted value, which means it is a heuristic used to approximate the expected future rewards. This is different from the reward function itself which gives an immediate sense of desirability of a state compared to what can be obtained in the long run.

Reinforcement learning tasks can further be broken down into episodic or continuing tasks. For example, con-

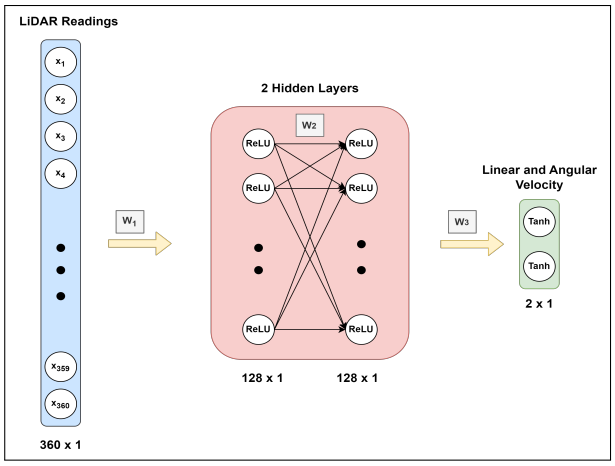

Figure 1. The neural network is parameterized by weights W1, W2 and W3. The input to the network are the 360 LiDAR readings. The network outputs the linear and angular velocity for the robot.

sider a robot which is to learn to escape a maze. Giving it a reward of +1 when it escapes a maze and 0 when it is not able to do so can be described as episodic. However, if we consider the case of cart-pole balance problem, the agent, at each time step, is given a reward of +1 for staying upright. This can be described as continuing task. In this project, the REINFORCE algorithm [11] will be implemented. The details on how this algorithm works is presented in section 2.4. Also, since we are dealing with a Robot control task, the implementation will deal with continuous action spaces and Gaussian exploration as described in [10].

3.1. Neural Network Architecture

The control policy for the robot is modelled using a fully-connected 3 layer perceptron as shown in Figure 1. To introduce non-linearity between layers, ReLU activation was chosen for the neurons in the hidden layers. Since the output velocity must be a value which can be scaled by the user, the activation for the output layer was chosen to be tanh to scale the value between -1 and 1.

3.2. Reward Function

The goal for the robot is to track a wall on its left and maintain a particular distance to the wall at all times. To generalize the task, the robot might also encounter sharp corners where the angle between the wall to its left and in its front is 90 degree. To generalize this task, the reward function must also capture the information provided by LiDAR readings in front of the robot.

To get a heuristic of whether the robot is within a particular distance to the wall on its left, the average distance to wall is calculated using LiDAR readings ranging from 88 degree to 92 degree as shown in Figure 2. Once the average distance is calculated, the required distance to wall

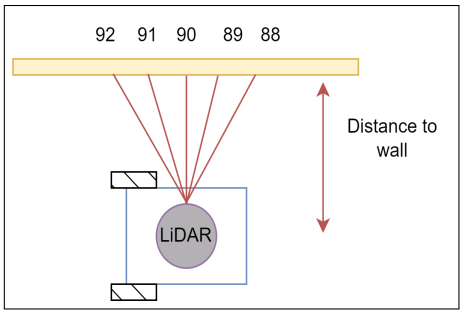

Figure 2. The LiDAR captures the average distance to wall on its left using the readings at 88 - 92 degrees.

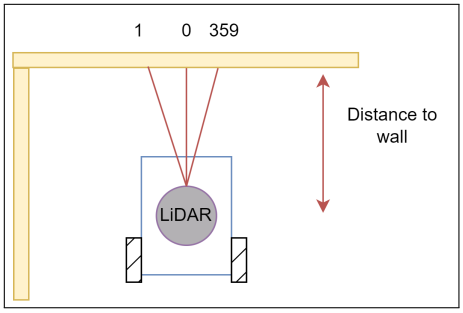

Figure 3. The robot captures the presence of a wall in front by using the LiDAR values at 1, 0 and 359 degree in its front.

(specified as hyperparameter) is subtracted from it and its absolute value is taken to obtain δ. δ is then compared with an allowed amount of error, ϵ (specified as a hyperparameter). If the absolute value is less than ϵ, then a positive reward of +1 is accumulated, else, a negative reward of -1 is accumulated.

For the case when the robot is facing a wall and needs to make a sharp turn, it uses the LiDAR readings in front of the robot; 1 degree, 0 degree and 359 degree as shown in Figure 3.

Based on the above measurements, the reward is calculated for the actions of setting the forward and angular velocity separately to provide the reinforce signal for back propagation. To calculate the reward for the forward velocity, the following function is used:

$$\mathcal{R}_{linear} = \mathcal{R}_{fwd} + \mathcal{R}_{front}$$

where,

$$\mathcal{R}_{fwd} = v_{fwd}$$

and v_{fwd} is the linear velocity of the robot.

$$\mathcal{R}_{front} = \begin{cases} +1 & \text{if two of the front lidar readings are more than allowed distance to wall} \\ -1 & \text{otherwise} \end{cases}$$

$\mathcal{R}_{fwd}$ is to incentivize forward motion of the robot while

$\mathcal{R}_{front}$ is to disincentivize forward collision with walls when making sharp turns. In case of angular velocity,

$$\mathcal{R}_{angular} = \begin{cases} +1 & \delta < \epsilon \\ -1 & \delta > \epsilon \end{cases}$$

where,

δ = average distance to wall - required distance

ϵ = allowed error bounds

For the scenarios when $\delta < \epsilon$ the robot is within the specified bounds and thus is performing the task successfully. However, whenever it goes out of bounds, the robot is penalized with a negative reward of -1.

3.3. Future Expected Reward

The goal of REINFORCE is to maximize the future expected reward starting from a state. To estimate this heuristic, we use the concept of cumulative reward with discounting, that is, future rewards are less worthwhile than immediate rewards. Thus, for our policy based learning, we calculate the value of any given state (s_i, at time step i) by using the following equation:

$$R(s_i) = \sum_{\tau=T}^{\tau=i} \gamma^{(T-\tau)} \cdot R_\tau \tag{1}$$

where, γ is the discount factor. γ is typically a value between 0 and 1.

3.4. On Policy Reinforcement Learning

As mentioned in introduction, value based methods are suitable for discrete action spaces where one is able to take an argmax over the finite set of action spaces. However, when dealing with continuous action spaces, policy based methods are ideal as it directly maps input to actions. This time, however, we will be estimating a probability distribution over an action space and thus, the method is stochastic. In our case, we assume for each input, the output distribution is a 1D gaussian with σ_{lin} and σ_{ang} as the standard deviation and μ_{lin} and μ_{ang} as the mean of the linear and angular velocity respectively. To start with, we initialize a policy π with parameters θ. Now, we sample trajectories using either simulation or real robot and if the trajectory was "good", increase the parameters θ towards that particular trajectory and decrease the parameters θ towards a trajectory which resulted in a negative/reduced reward. Assuming an action a^* was good, we have the following update rule:

$$\theta_{t+1} = \theta_t + \alpha \cdot \Delta\pi_{\theta_t}(a^*|s) \tag{2}$$

Now, we need an estimate as to how "good" a trajectory is. Here, we will be using the standard discounted reward heuristic as mentioned above. Thus, the update rule now becomes:

$$\theta_{t+1} = \theta_t + \alpha \cdot R(s,a)\Delta\pi_{\theta_t}(a|s) \tag{3}$$

However, since $\pi_{\theta_t}(a|s)$ defines the probability, we will end up sampling more of such "good" actions and might end up with instability due to doubly moving in the direction of gradient. Thus, we weight each update with its probability giving us the final gradient ascent rule which is key to the REINFORCE algorithm as defined by Williams in 1992 [11]:

$$\theta_{t+1} = \theta_t + \alpha \cdot R(s,a)\frac{\Delta\pi_{\theta_t}(a|s)}{\pi_{\theta_t}(a|s)} \tag{4}$$

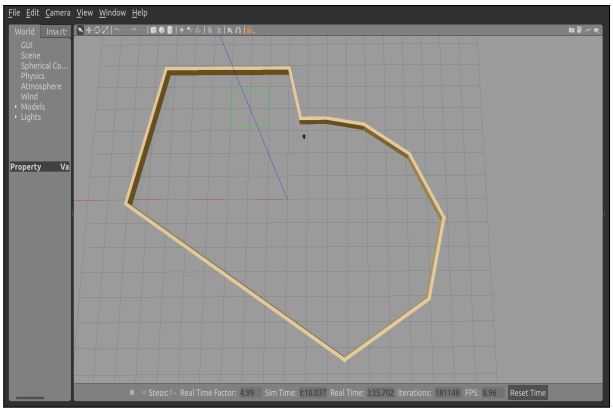

Figure 4. The training world consists of a closed loop of walls with each making sharper turn with respect to the previous one.

4. Simulation Environment and Training

Gazebo along with ROS packages for Turtlebot3 were used to train the nominal control policy. Figure 4 shows the training environment in Gazebo.

Starting from the location where the robot is located in Figure 4, the walls are angled in 10 degree increments. For example, the second wall makes a 10 degree angle with the first wall; the third wall makes 20 degree angle with the second wall and so on. The intuition behind this is that the robot must learn to make all acute angle turns. If the robot is able to complete the full track, it tells us whether the neural network policy was able to generalize well.

Figure 5 gives a close up of the robot model used. It houses a 2D 360 degree lidar on top. These measurements are used as input to the neural network and determining how far away the walls are in its front and left. The LiDAR has a measurement range of 0.12 - 3.5 meters. Thus, before feeding the LiDAR measurements to the neural network, the measurements are clipped to between 0.15 - 3.00 and normalised from 0 - 1 to provide consistent data input to the network for faster learning.

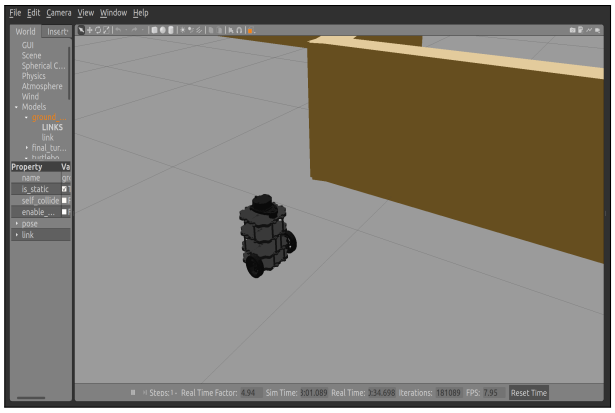

Figure 5. The Turtlebot3 burger model used in simulation.

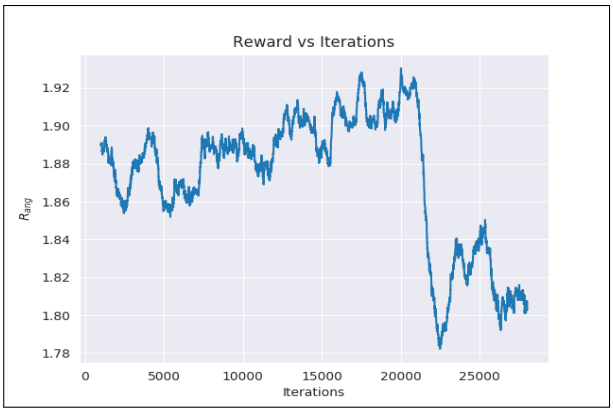

Figure 6. The average reward accumulated with each episode, for the action of setting angular velocity.

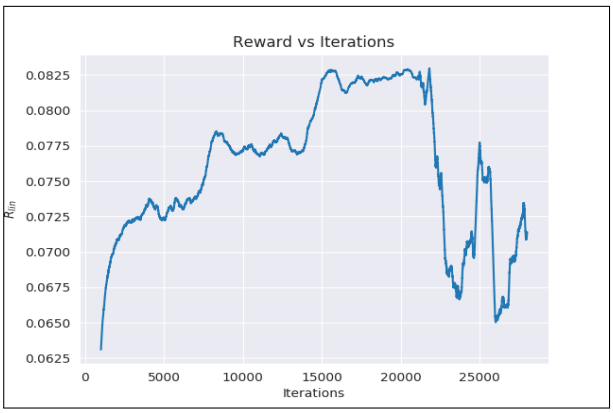

Figure 7. The average reward accumulated with each episode, for the action of setting linear velocity.

To speed up the training process, the update factor for Gazebo was set such that it offered a consistent 5x speed up with respect to real time. This also meant that the sample rate of the LiDAR had to be set to 5 times to that in the real world, that is, it was increased from 5 to 25 Hz. With regards to performing weight update. The basic pseudo code for the algorithm is given as follows:

Algorithm 1 REINFORCE

Require: $\pi_{\theta_0}(s), R(s_{t+1}, a_t), S, T(s_{t+1}|s_t, a_t), \sigma$
  for i = 0 to 28000 **do**
    $P \leftarrow <>$
    $R \leftarrow <>$
    $s_0 \sim S$
    for t = 0 to T **do**
      $\mu_t \leftarrow \pi_{\theta_i}(s_t)$ ▷ Mean of Gaussian
      $a_t \sim \mathcal{N}(\mu_t, \sigma)$ ▷ Sample action
      $s_{t+1} \sim T(\cdot|s_t, a_t)$ ▷ Execute in simulation
      $r_t \leftarrow R(s_{t+1}, a_t)$ ▷ Collect reward
      $R \leftarrow R \cup r_t$
      $P \leftarrow P \cup \mathcal{N}(a_t, \sigma)$ ▷ Prob of a_t
    end for
    for r_t in R **do**
      $r_t \leftarrow \sum_{\tau=t}^{\tau=T} \gamma^{\tau-t} \cdot r_t$ ▷ Cummulative reward
    end for
    $\theta_{i+1} \leftarrow \theta_i + \alpha \cdot r_t \cdot \sum_{t=0}^{t=T} \nabla_\theta log(P_t)$ ▷ NN update
  end for

The control policy was trained for 28000 episodes with a learning rate of 0.0001. The standard deviation for estimating the gaussian output of the neural network was set to 0.475 and 0.0475 for angular velocity and linear velocity respectively. As training iterations increase, this value for sigma is reduced since less and less exploration is required as learning takes place. In particular, after every 7000 iterations this value is reduced by 0.125 and 0.0125 until it reaches a value of 0.1 and 0.01 for angular and linear velocity respectively.

4.1. Results

To infer how the training has progressed, plots for the average rewards accumulated for each episode and episode length vs iterations were obtained. Plots for weight change for each layer with respect to iteration were also used to get a better idea of how gradients were changing with each iteration. Since the acutal data is noisy, a moving average of 1000 values was used to plot the data.

As can be seen from Figure 6, the general trend is an increase in rewards which is to be expected. As the robot learns better actions with each iteration, the average reward accumulated in each iteration increases.

A similar trend as seen in Figure 6 is obtained in Figure 7 which contains the data for average reward for linear velocity with iteration. The curve in this case is a lot smoother because the reward for linear velocity is a smooth function which depends on the linear velocity and not a discrete function as in the case of taking angular turns.

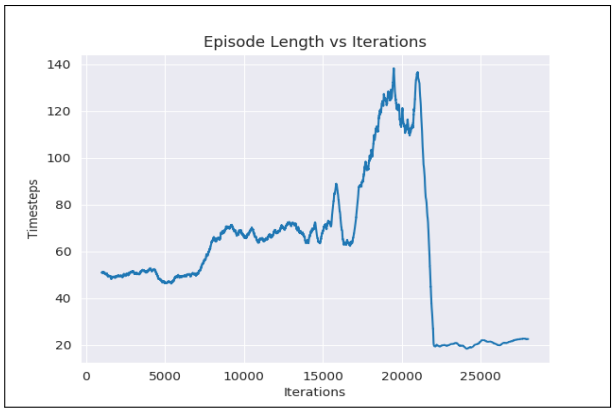

Figure 8. Episode length vs iteration

Figure 9. Sum of absolute value for gradients for each layer weight terms.

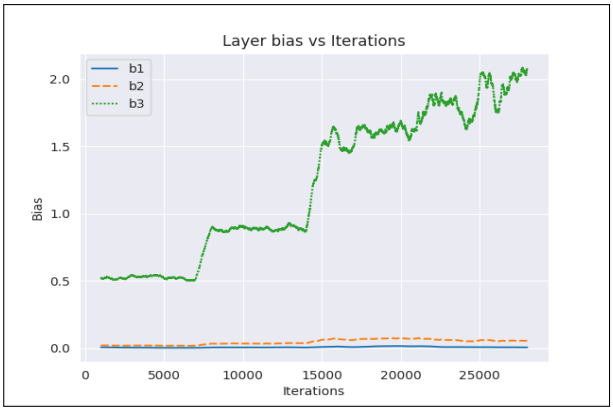

Figure 10. Sum of absolute value for gradients for each layer of bias terms.

Figure 8 shows the episode length with iterations. Again, this increasing trend was to be expected as the policy improves, the robot is able to sample from better actions and thus stay inside the error bounds for increasing intervals of time. However, somewhere between 21000 and 22000 iteration, the plot takes a shapr dip in episode length and reaches an all time minimum of 20 episode.

A trend common to each of these plots is that there is sharp dip at around 21000 iterations. This was unexpected as the general trend should have been increasing. One of the possible explanations is a low value for sigma. If one looks at the formula for the normal distribution, it reaches a limit when sigma tends towards zero.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}(\frac{x-\mu}{\sigma})^2} \tag{5}$$

Thus, the gradient update might be unstable for very low values of sigma which might have caused the training to crash at around 21000 iterations which is when sigma is set to 0.1 and 0.01 for angular and linear velocity respectively. To further verify this, the gradients were accumulated for each iteration and the following trend was observed:

In Figure 9 and 10, although the gradients do increase to a certain extent at around 21000, the values are not arbitrarily high or random. Thus, even though the weight update rule might be consistent, the high value of gradient might be the reason for the crash in training. To conclude, it seems that the training is highly sensitive to the value of sigma that one chooses and thus it is essential to check for what values of sigma the training becomes noisy or inconsistent.

As one might guess from the earlier plots, the best policy might be the one obtained at around 21000 iteration. After verifying the performance for the policies obtained between 20000 and 21000, the best policy was obtained at 20500 iteration number. The performance of this policy with respect to the training environment can be seen in Figure 11. The first subplot in Figure 11 shows how the robot is moving with respect to the wall on its left. In particular, it plots the 90 degree lidar distance measurement with each timestep. The red band represents the error bound which is +/- 0.1 m for our experiemnt. The seconds subplot gives the percentage of time spent inside and outside the error range while completing the entire loop.

While training the control policy, the maximum episode length was set to 500. However, the robot is still able to complete the entire track which is around 2300 timesteps without colliding with any of the walls. This shows that the training was able to generalize well even in the case of making sharp 80 degree turns.

5. Learning to Learn

Now that we have a nominally trained control policy, this section investigates online update of a neural network policy while the robot is being deployed. The motivation behind this is that while deploying any real world system, it must account for disturbance or change of dynamic model as the

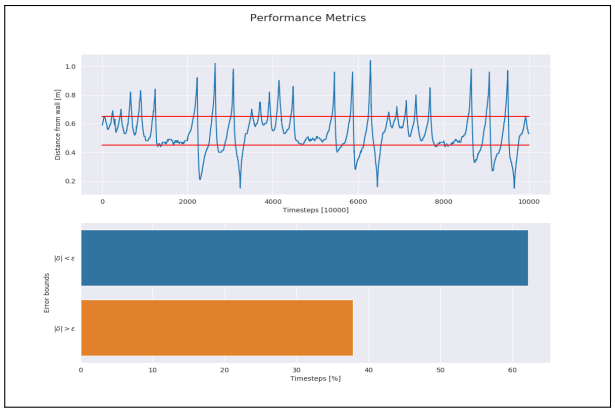

Figure 11. Performance metrics for the best nominal policy achieved after 28000 iterations.

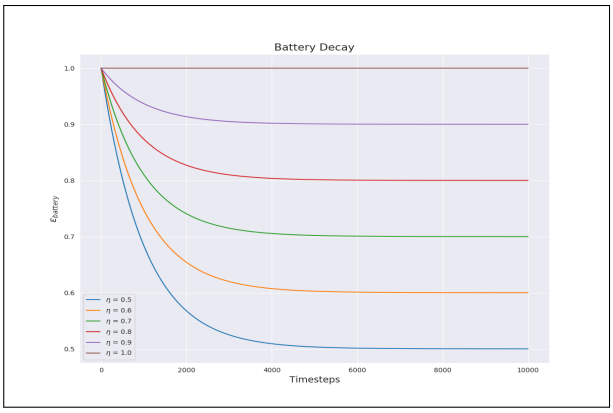

Figure 12. Exponential battery decay model.

environment is constantly changing. Thus, if we can gather data online and update our policy in real time it can continuously improve and account for any change of physics related to the system or environment. One of the challenges to performing real time update is access to limited information. Thus the method that one uses must be data-efficient. Another challenge is to determine how long to wait before performing gradient update. If we choose to update the network at every instant, it might result in a noisy update whereas if we wait long periods of time to gain as much information as possible, the model might overfit to that particular update and might also increase the real time complexity of the algorithm. In the case of reinforcement learning, reward shaping can be used to provide a small and steady signal to move in a favorable direction. Reward shaping is setting up a continuous reward function which gives the agent a reward based on the state action pair. Now, since we already have a nominal policy this can be particularly helpful for performing online update as it provides faster convergence than a discrete reward function. However one must be careful not to over tune the policy as well because it might then loose the information that it started out with. To combat this, we use the technique of transfer learning. In order to provide flexibility, we update the weights only for the final layer and freeze the weights of the layers before it. This ensures that the neural network always encapsulates a general idea of control while also allowing for flexibility to change in system or environment dynamics.

5.1. Change in Dynamics

One way which we are going to emulate the change in system dynamics is by assuming that the battery levels are changing and as a result of that, the actuator output is depreciating with time. For example in simulation, while sending an output command of moving with 0.1 m/sec, the actual output will be a factor $\epsilon_{battery}$ times 0.1 m/sec. For simplicity we will be assuming that the battery levels only affect the angular velocity of the robot and not the linear velocity. To model the battery decay for the robot, we use an exponentially decaying model given by the following equation:

$$\epsilon_{battery} = \eta + (1 - \eta) \cdot e^{\frac{-t}{1000}} \tag{6}$$

The curve for the battery decay model is shown in Figure 12. The curve starts with an $\epsilon_{battery}$ value of 1 and exponentially decays to a final value determined by the parameter η. The reason behind choosing this model was that when initially the battery is fully charged, it offers high performance with output nearly equal to the desired value. However, as operation time increases, the value quickly drops to a nominal value of say η in our case. In our experiments, the value for epsilon will be multiplied to the output angular velocity command given by the control policy.

5.2. Reward Shaping

The scheme for the online update is the same as for training with the difference of the reward function. The reward function used in this case is a continuous function as follows:

$$\mathcal{R}_{angular} = \begin{cases} \delta & |\delta| < \epsilon \\ \delta & \delta < 0 \\ -\delta & \delta > 0 \end{cases}$$

where,

$\delta =$ Distance to wall - Required distance to wall

$\epsilon =$ Error bounds

The intuition behind this reward function is as follows. For the case when the robot is within the error bounds, that is, $\delta < \epsilon$, we provide a positive reward which is maximum

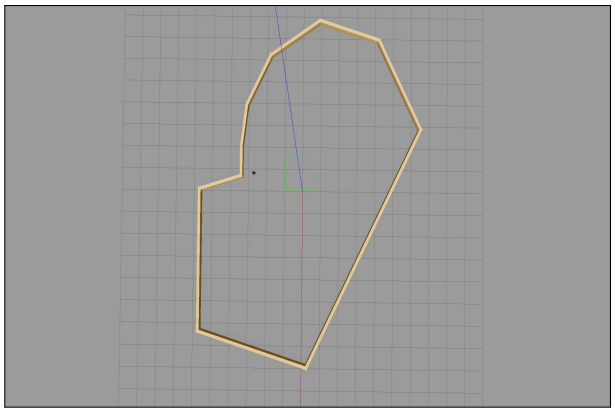

Figure 13. Environment used for testing nominal and stochastic policy with online update.

Figure 14. Nominal policy with $\eta = 0.6$ (best performing sample).

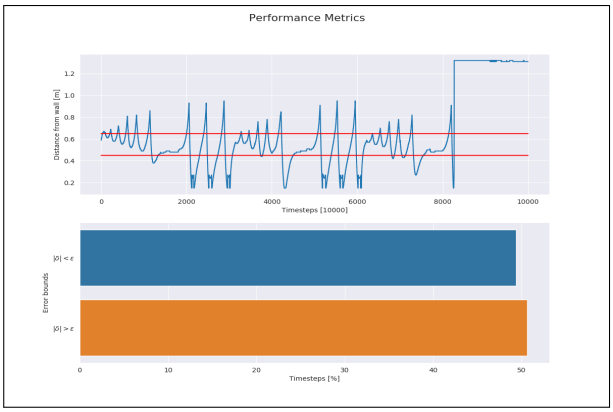

Figure 15. Nominal policy with $\eta = 0.5$ (best performing sample). The sharp vertical line at 8000 timesteps and then a flat horizontal line represents the robot crashed into the wall and could not make it any further.

when the robot is at the exact prescribed distance and reduces at it approaches the boundaries. For when the robot is outside the boundaries, we provide negative rewards equal to δ when it is less than the lower boundary and -δ when it is more than the upper boundary. This ensures that we provide the robot with some incentive at each time step to stay close to the prescribed distance. The reward for the linear velocity update is kept the same.

5.3. Results

As can be seen in Figure 12, 6 battery decay models were tested. Firstly, the nominal policy was tested, in which the model weights were not actively updated. Surprisingly, the nominal policy is able to complete 100000 timesteps in simulation without crashing into the wall for η values from 0.6 to 1.0. with an increment of 0.1. Although the robot does not crash into the wall, it does graze the wall at times. The gazebo world used for this first set of experiments is shown in Figure 13. One of the sample plots is shown in Figure 14. Please refer to Appendix A for the full set of experimental plots.

For the case of $\eta = 0.5$ however, the model seems to perform poorly, with the worst case being able to complete 2000 timesteps and best case being 8000 timesteps before colliding into the wall. One of the best cases out of 10 trials is shown in Figure 14.

To test the online update of weights, the neural network weights of all layers except the last one are frozen. This can be thought of as similar to having a higher level and low level controller. The nominal policy with frozen weights represents the high level controller, containing the general idea of what the control policy should look like in general. Whereas the final layer, can be thought of as representing a low level controller which can cater to different situations as time evolves, for instance, decrease in actuator output due to battery decay in our case or any other change in dynamics that can take place over a period of time. Also, while deploying this model, the standard deviation is set to 0.1 for angular velocity and 0.01 for linear velocity. This value is low enough so that the model does not stray too far away from the nominal policy but also sufficient to provide some form of exploration. The learning rate is also increased to 0.1 versus 0.0001 which was used in training.

The stochastic policy with online update performs worse than the deterministic nominal policy for η values from 0.6 to 1.0. As can be seen from the plots in Appendix B, the policy is able to complete the loop at times but is not as consistent as the nominal policy. However, for $\eta = 0.5$, the stochastic policy performs marginally better than the nominal policy with the worst case being the same as the nominal policy. However, the best case is when the robot is able to complete the 100000 timesteps. This result is shown in Figure 16.

The active update of weights thus proves to be beneficial for the worst case of battery decay model being con-

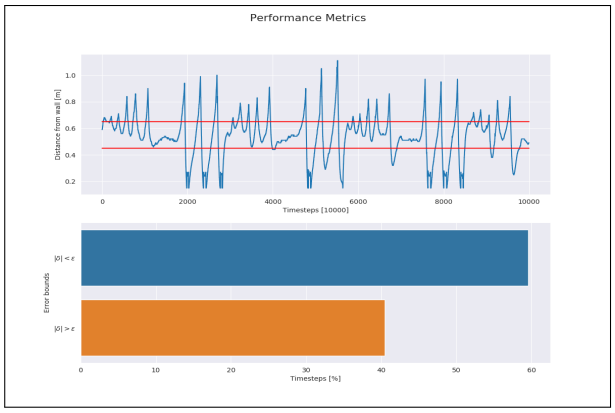

Figure 16. Stochastic policy with online update and $\eta = 0.5$ (best performing sample). In this sample the robot achieves better performance than the nominal policy by completing the entire 10000 timesteps without crashing into the wall.

Figure 17. Turtlebot3 Burger model with Raspberry Pi 4B and 2GB RAM

sidered with $\eta = 0.5$. With this, we can conclude that it is worthwhile to deploy a stochastic policy particularly in areas where there is high amount of uncertainty. Thus, with active learning, it is more likely that the robot would end up finding a better policy than a normal one in uncertain environments.

6. Implementation in Real World

Figures 17 and 18 shows the Turtlebot3 robot used and the environment setup. The goal for the robot in this setup is to track the box on its left.

One modification that had to be made for the real world was LiDAR pre-processing before feeding it into the network. The LiDAR data in the real world was not as consistent and would not provide with 360 measurements every time. It would provide somewhere in the range of 220 to 235 number of measurements around the robot. Thus, it was necessary to perform a co-ordinate transformation and

Figure 18. Envrionment setup for real world implementation

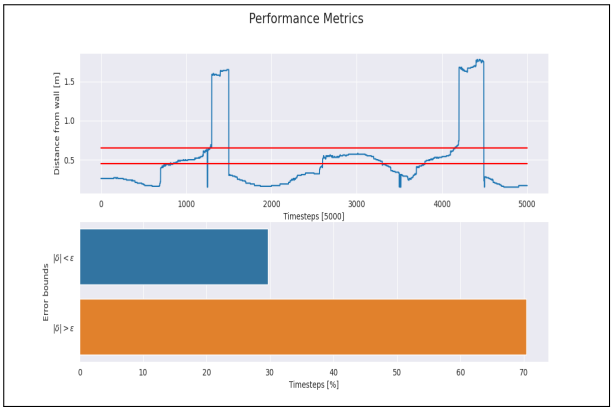

Figure 19. Nominal policy without online update deployed in real world

interpolation for less number of measurements (say 220) spread across 360 degrees to 360 measurements across 360 degrees. Now, the input to neural network is consistent even when the number of LiDAR readings were to change. With this, the nominal and stochastic policy were deployed in the real world, with the same neural network and parameters as those used in simulation. Figure 19 and 20 show the results from implementing the nominal policy and stochastic policy in the real world.

From the plots in figures 19 and 20 we can see that the performance of the nominal and stochastic policy is similar and not very good at that. This is majorly due to the fact that the neural network model is receiving lesser information that what it received during training (360 LiDAR measurements). However, the behaviour in the real world is very close to that observed in simulation given this critical disparity between training and real-time deployment.

7. Conclusion

This work involved the implementation of REINFORCE algorithm to learn a model free neural network control pol-

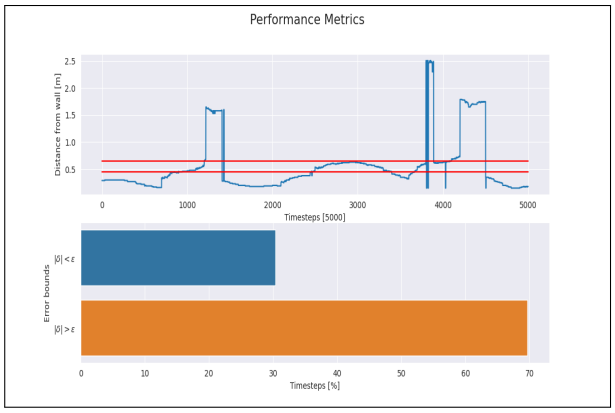

Figure 20. Stochastic policy with online update deployed in real world

icy on continuous action spaces from scratch. This work also investigated performing an online update of the neural network to account for changing dynamics. The control task chosen for this work was wall tracking by a modular two-wheeled robot Turtlebot3 burger model. The goal of the control task is to continuously move the robot while maintaining a specified distance to the wall. The nominal policy was trained in Gazebo simulator using ROS packages and python. Gaussian exploration was used to sample from continuous action space. For the stochastic policy with online update, all layers of the neural network were frozen except for the last layer. This was to retain some aspects of the nominal policy while also allowing for flexibility while updating the neural network. Reward shaping along with gaussian exploration was used to learn about the changing environment. As a method of modelling uncertainty in environment, we assumed an exponential battery decay model and that the actuator output was linearly proportional to battery levels. Metrics were devised to give a sense of performance of policies. The nominal policy worked surprisingly well even in the presence of aggressively changing dynamics. However, the policy was not able to maintain the performance in the most adverse case. This was also where the stochastic policy beat the nominal policy in performance, although just barely. From this, we can conclude that the nominal policy performs well until the dynamics are really uncertain with respect to what it was trained for. Thus, using a stochastic policy with online update would shine the brightest when there are high levels of uncertainty in the environment. Finally, the nominal and stochastic policies were deployed in real-time. The control policy behaved in a similar manner as in simulation, however, due to lack of sufficient information from LiDAR (less than 360 measurement readings), the model did not perform as well as in simulation. Refer [6] for the github repository for this project.

References

[1] Pierre Yves Glorennec. Reinforcement learning: An overview. In *Proceedings European Symposium on Intelligent Techniques (ESIT-00), Aachen, Germany*, pages 14–15. Citeseer, 2000. 1

[2] Jamshed Iqbal, Mukhtar Ullah, Said Ghani Khan, Baizid Khelifa, and Saša Ćuković. Nonlinear control systems - a brief overview of historical and recent advances. *Nonlinear Engineering*, 6(4):301–312, 2017. 1

[3] Nicolas Minorsky. Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309, 1922. 1

[4] Aidan O'dwyer. *Handbook of PI and PID controller tuning rules*. World Scientific, 2009. 1

[5] Elena Pashenkova, Irina Rish, and Rina Dechter. Value iteration and policy iteration algorithms for markov decision problem. In *AAAI'96: workshop on structural issues in planning and temporal reasoning*, 1996. 1

[6] Manan Patel. Control policy using deep reinforcement learning. https://github.com/m-a-c-e/robo_capstone, 2022-2023. 10

[7] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. Online deep learning: Learning deep neural networks on the fly, 2017. 2

[8] Nahum Shimkin. *Nonlinear Control Systems*, pages 2886–2889. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. 1

[9] Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991. 1

[10] Hado van Hasselt and Marco A. Wiering. Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279, 2007. 2, 3

[11] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. 3, 4

[12] Dong-Ok Won, Klaus-Robert Müller, and Seong-Whan Lee. An adaptive deep reinforcement learning framework enables curling robots with human-like performance in real-world conditions. *Science Robotics*, 5(46):eabb9764, 2020. 2

A. Appendix

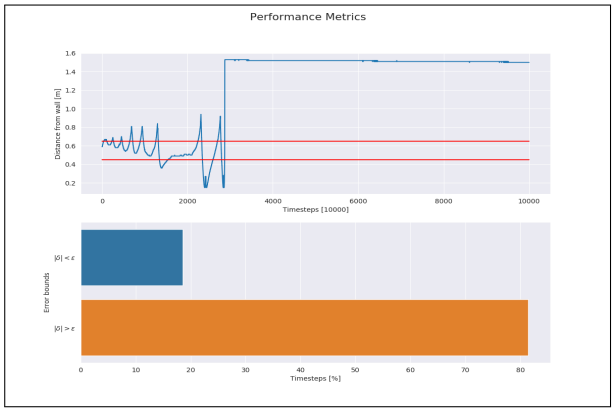

Figure 21. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 1)

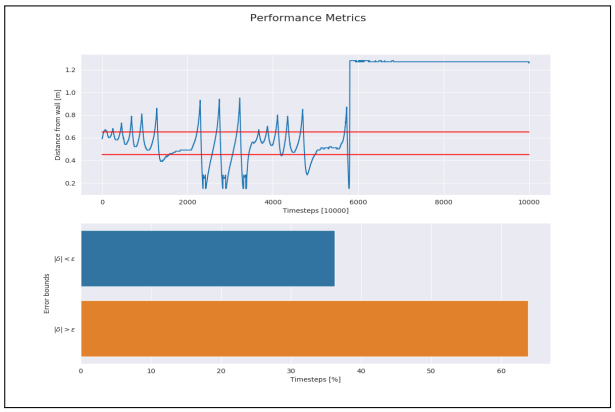

Figure 22. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 2)

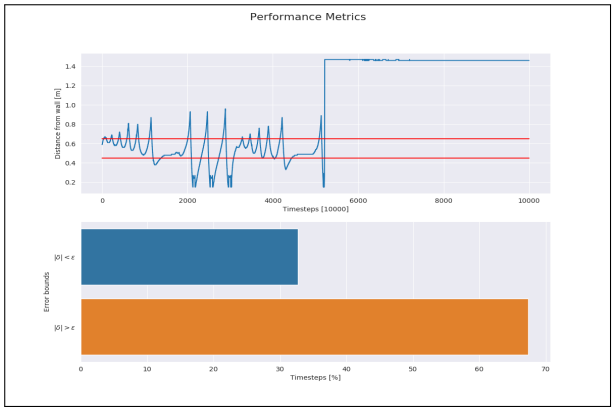

Figure 23. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 3)

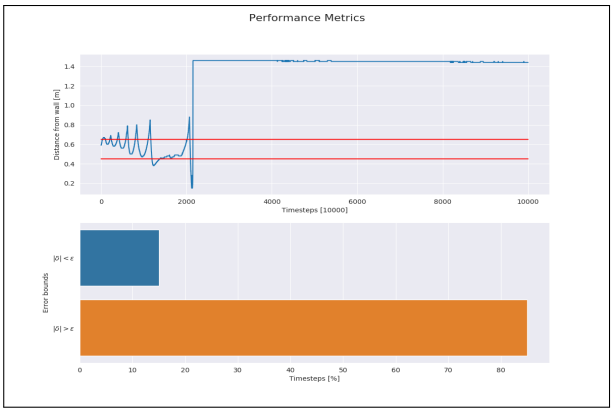

Figure 24. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 4)

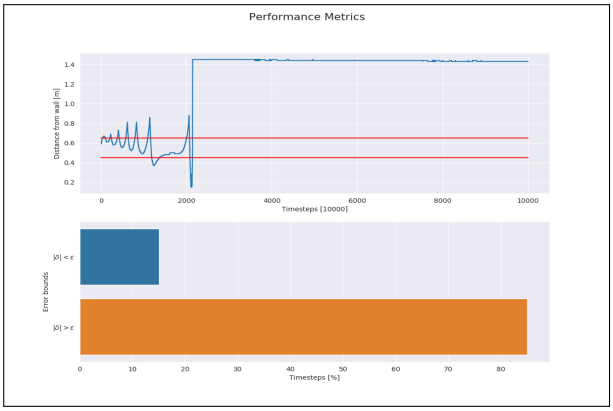

Figure 25. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 5)

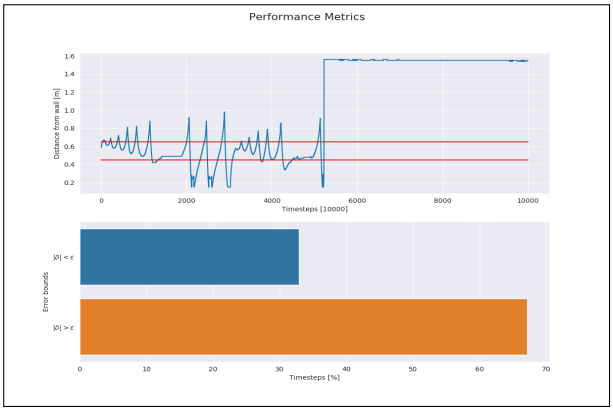

Figure 26. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 6)

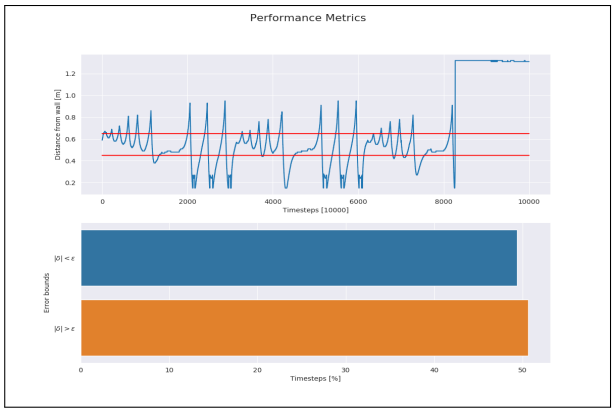

Figure 27. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 7)

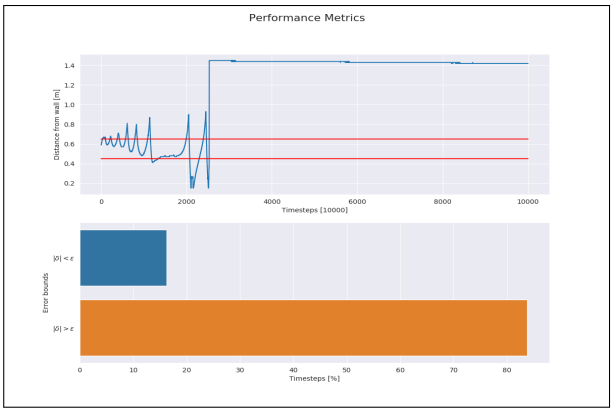

Figure 30. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 10)

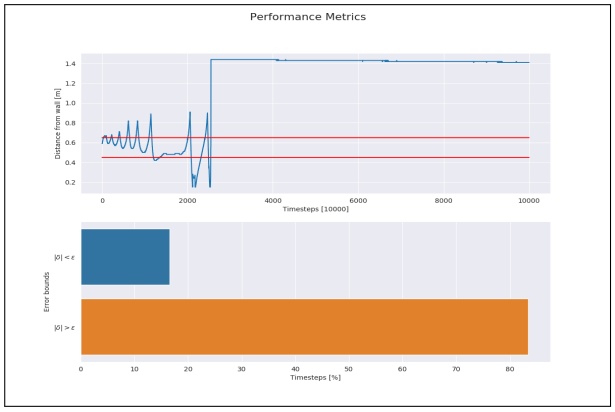

Figure 28. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 8)

Figure 31. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 1)

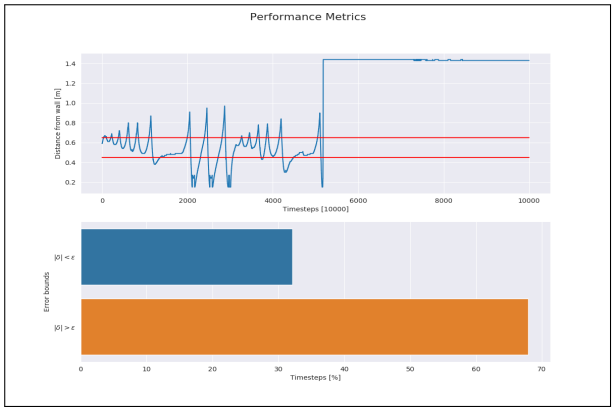

Figure 29. Nominal Policy without online update in custom world(1) ($\eta = 0.5$; sample 9)

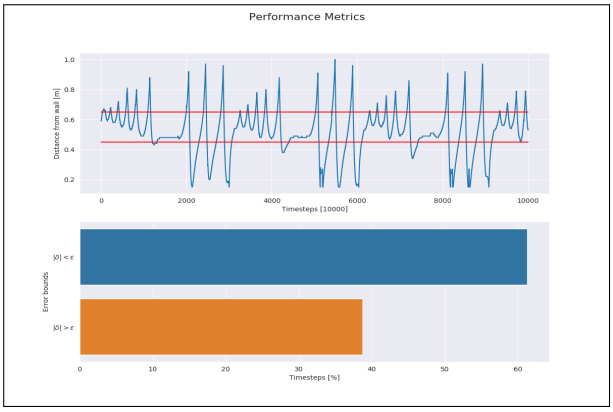

Figure 32. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 2)

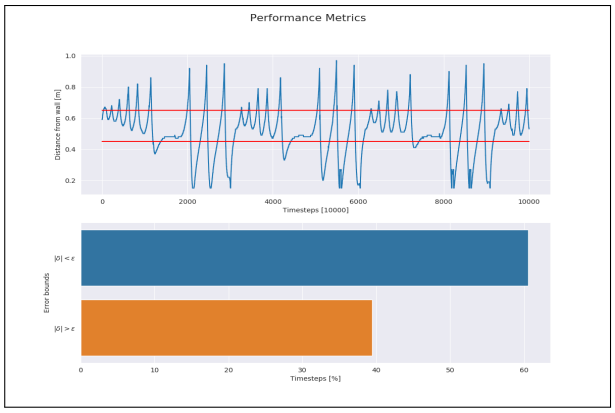

Figure 33. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 3)

Figure 34. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 4)

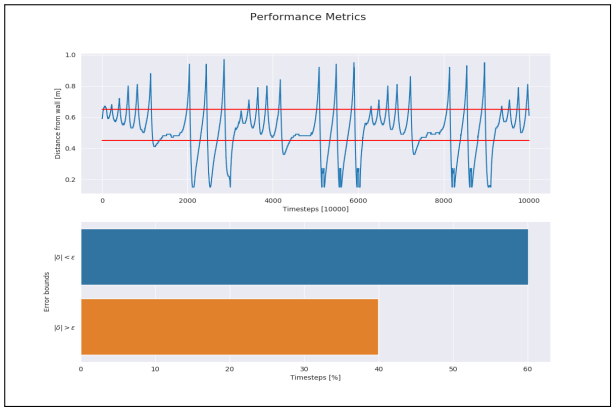

Figure 35. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 5)

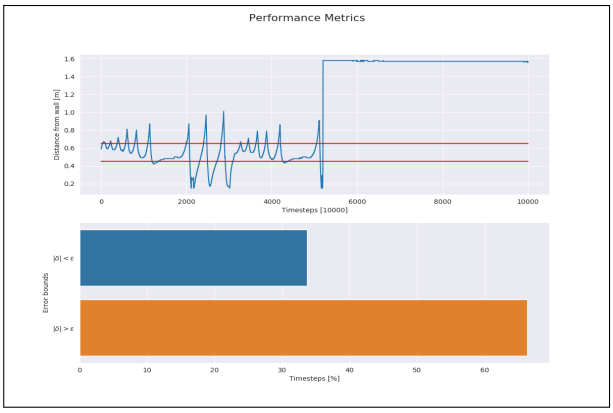

Figure 36. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 6)

Figure 37. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 7)

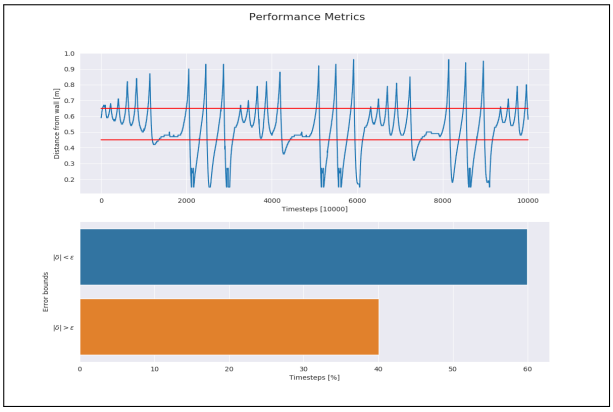

Figure 38. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 8)

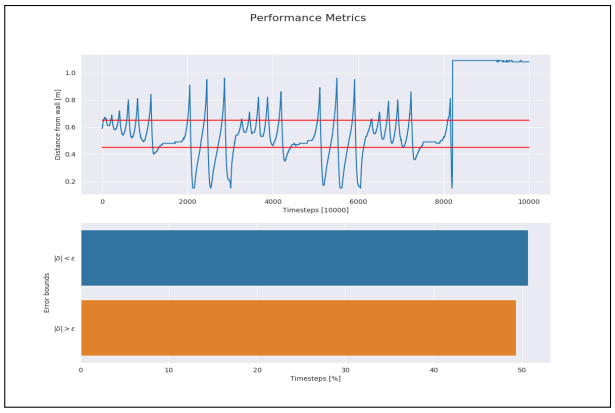

Figure 39. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 9)

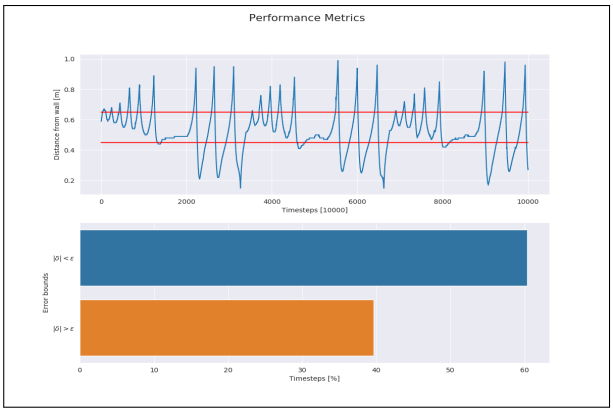

Figure 42. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 2)

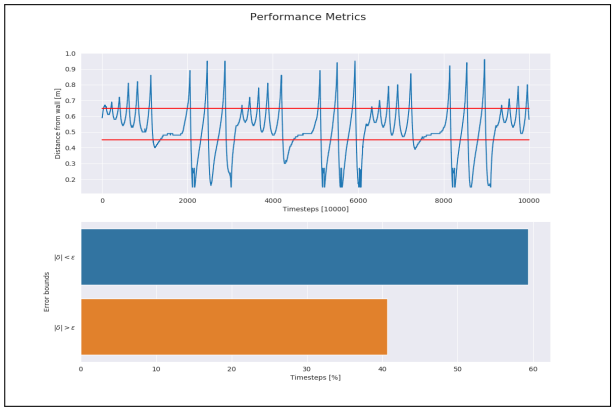

Figure 40. Nominal Policy without online update in custom world(1) ($\eta = 0.6$; sample 10)

Figure 43. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 3)

Figure 41. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 1)

Figure 44. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 4)

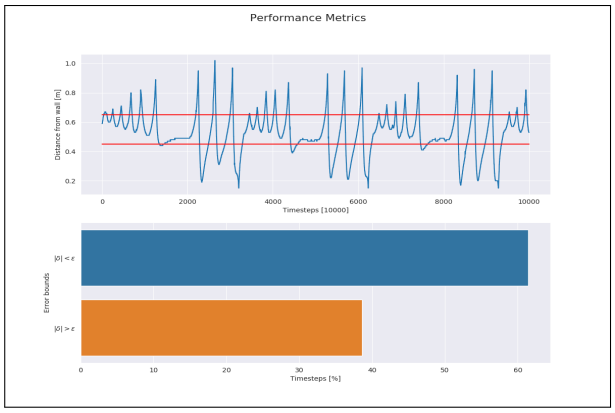

Figure 45. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 5)

Figure 48. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 8)

Figure 46. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 6)

Figure 49. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 9)

Figure 47. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 7)

Figure 50. Nominal Policy without online update in custom world(1) ($\eta = 0.7$; sample 10)

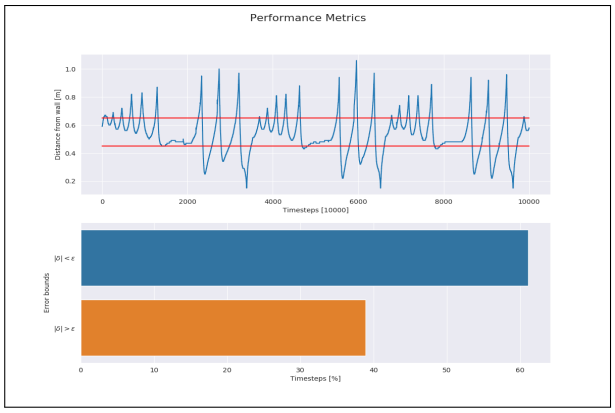

Figure 51. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 1)

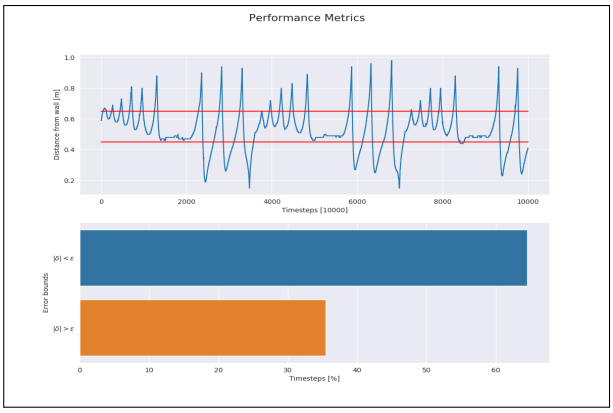

Figure 54. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 4)

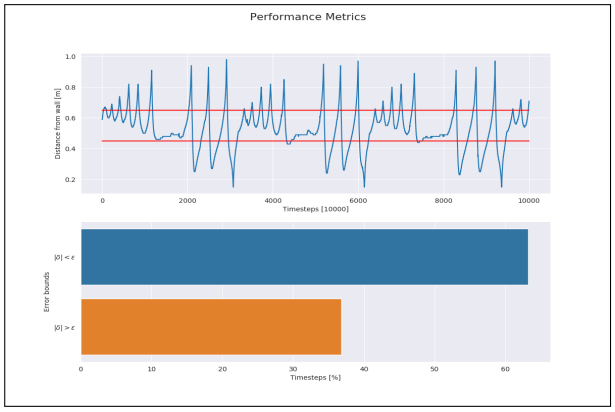

Figure 52. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 2)

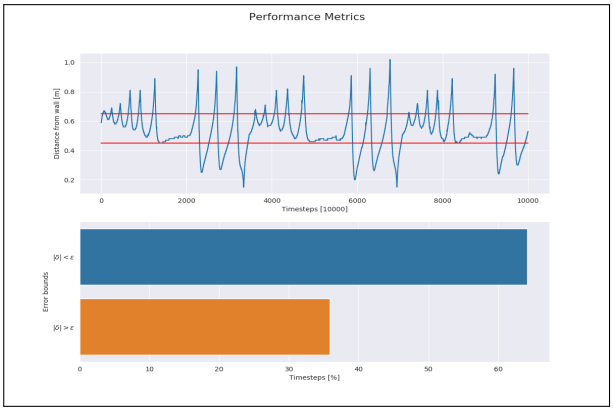

Figure 55. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 5)

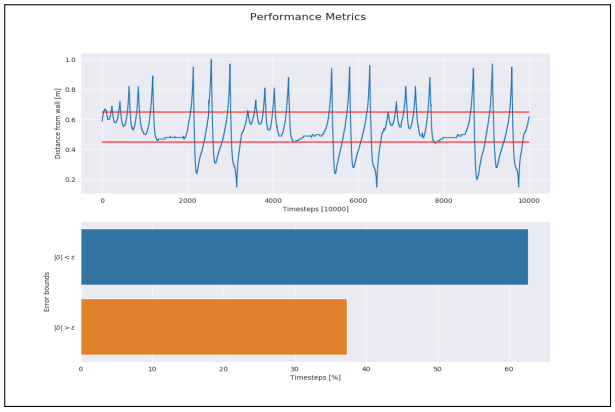

Figure 53. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 3)

Figure 56. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 6)

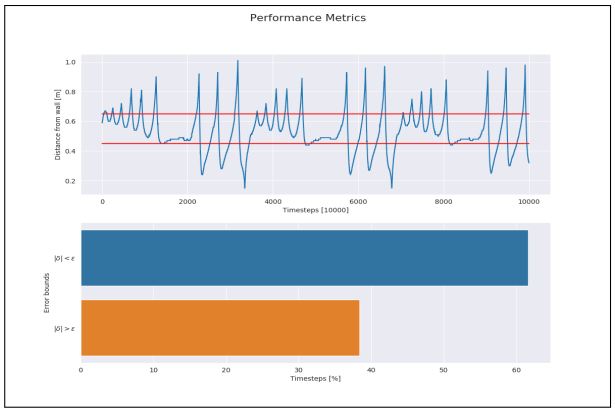

Figure 57. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 7)

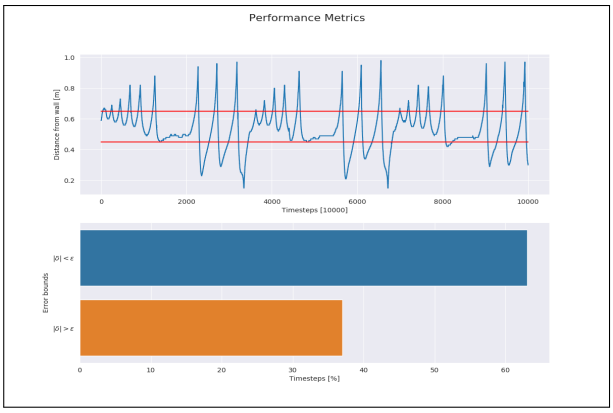

Figure 60. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 10)

Figure 58. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 8)

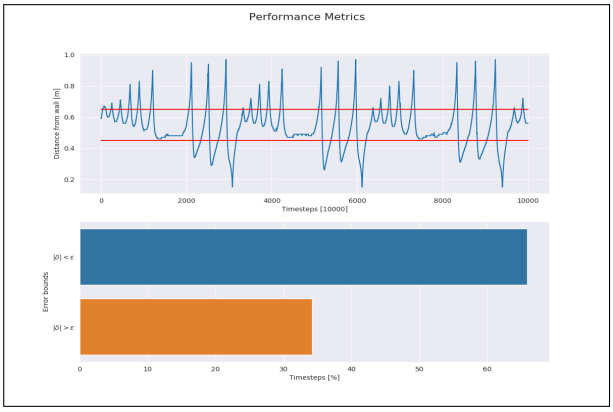

Figure 61. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 1)

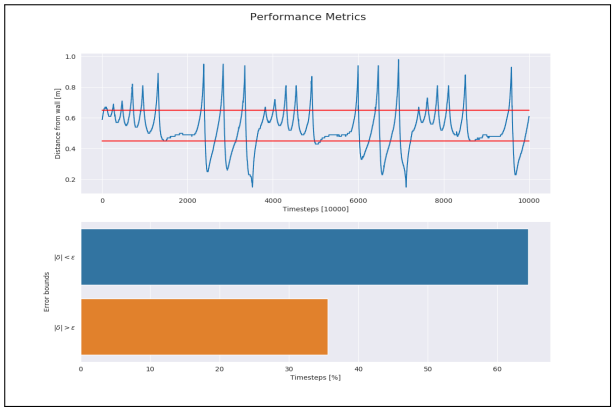

Figure 59. Nominal Policy without online update in custom world(1) ($\eta = 0.8$; sample 9)

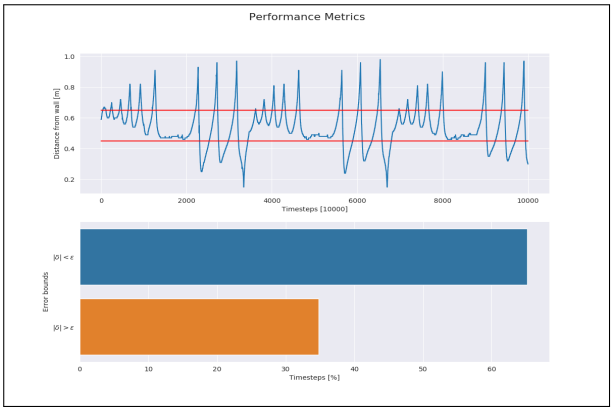

Figure 62. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 2)

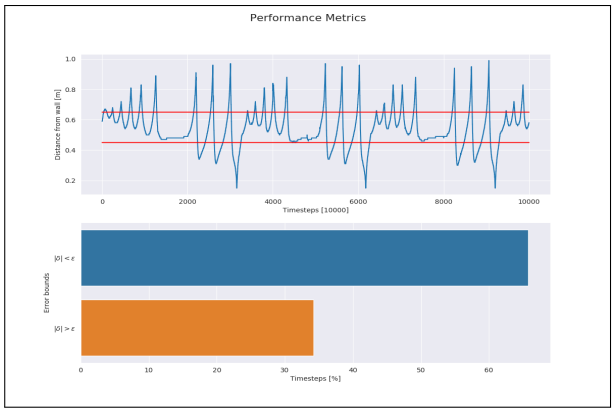

Figure 63. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 3)

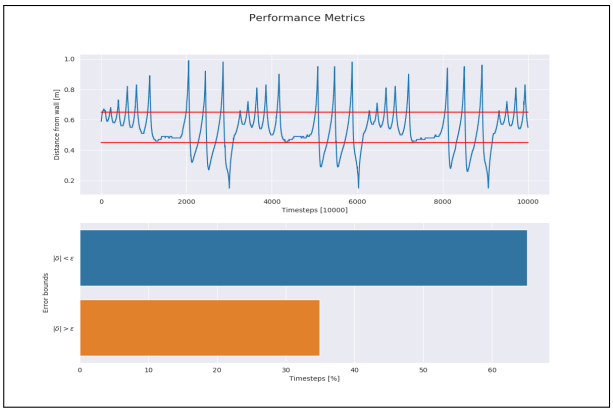

Figure 66. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 6)

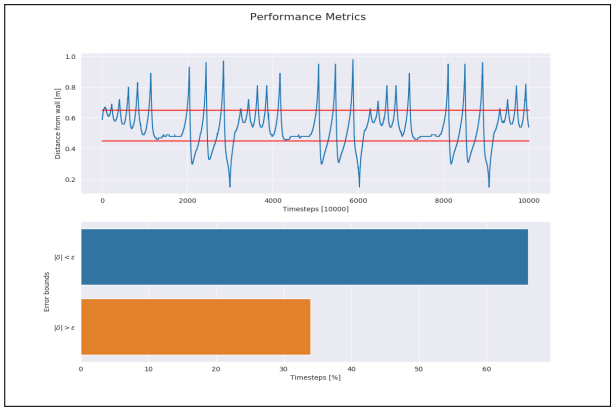

Figure 64. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 4)

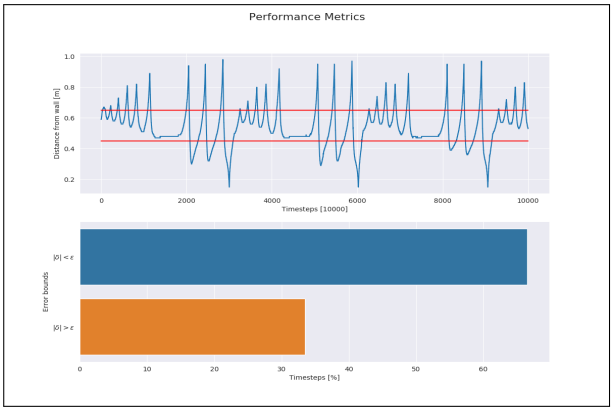

Figure 67. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 7)

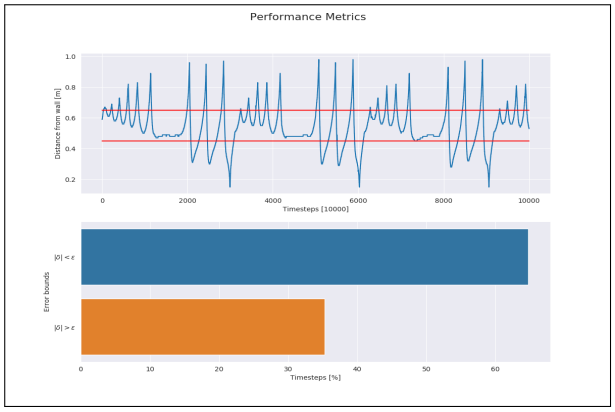

Figure 65. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 5)

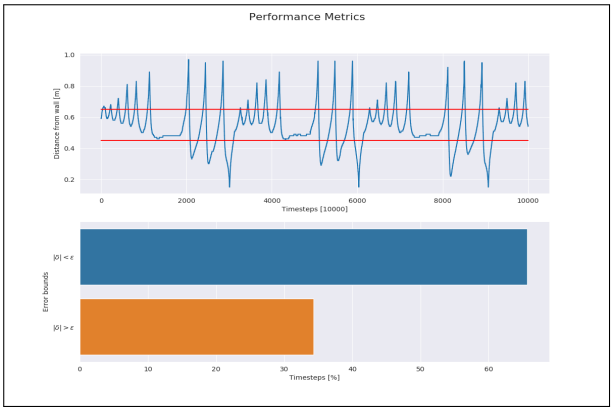

Figure 68. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 8)

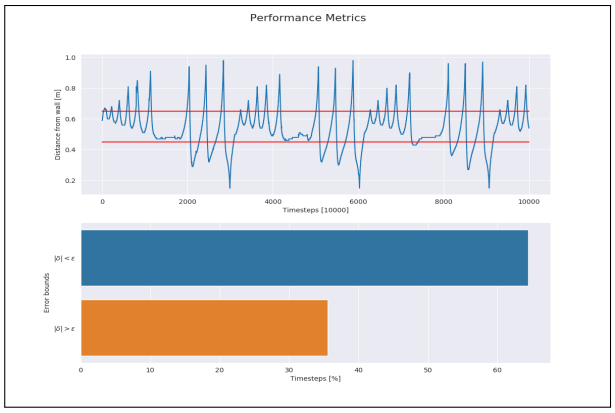

Figure 69. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 9)

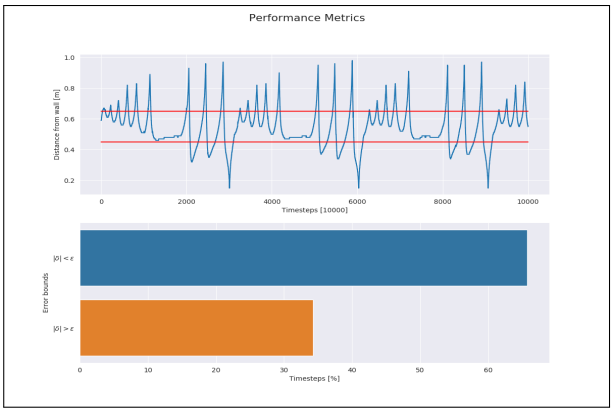

Figure 72. Nominal Policy without online update in custom world(1) ($\eta = 1.0$; sample 2)

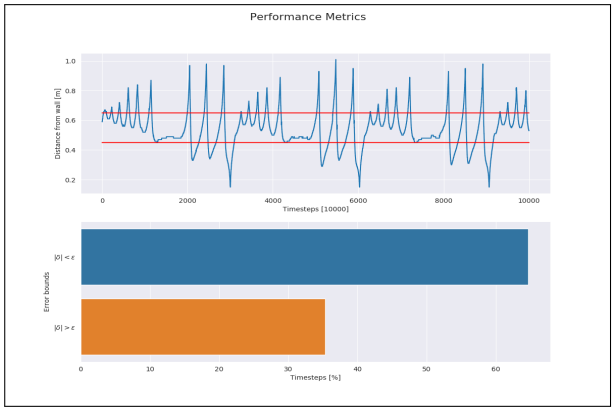

Figure 70. Nominal Policy without online update in custom world(1) ($\eta = 0.9$; sample 10)

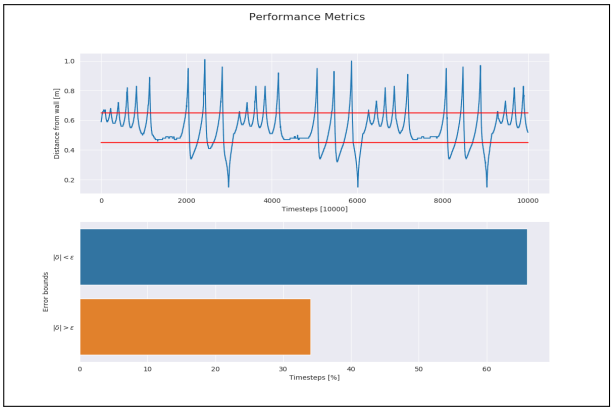

Figure 73. Nominal Policy without online update in custom world(1) ($\eta = 1.0$; sample 3)

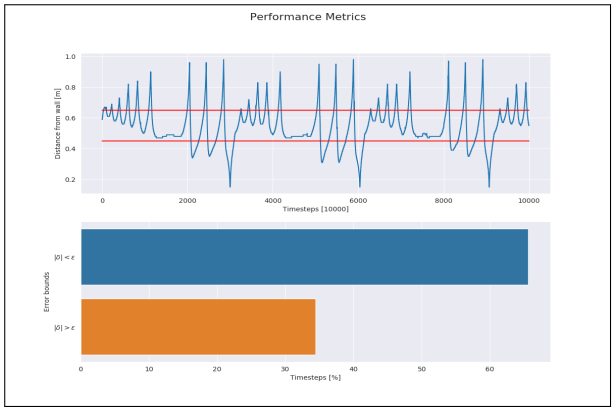

Figure 71. Nominal Policy without online update in custom world(1) ($\eta = 1.0$; sample 1)

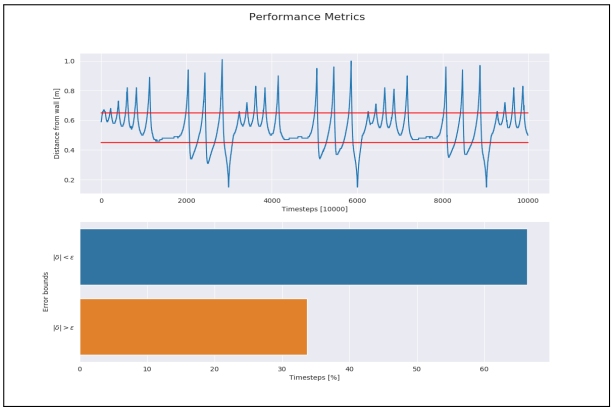

Figure 74. Nominal Policy without online update in custom world(1) ($\eta = 1.0$; sample 4)

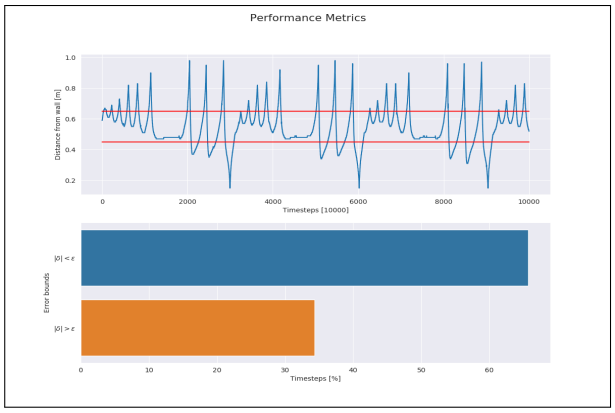

Figure 75. Nominal Policy without online update in custom world(1) (η = 1.0; sample 5)

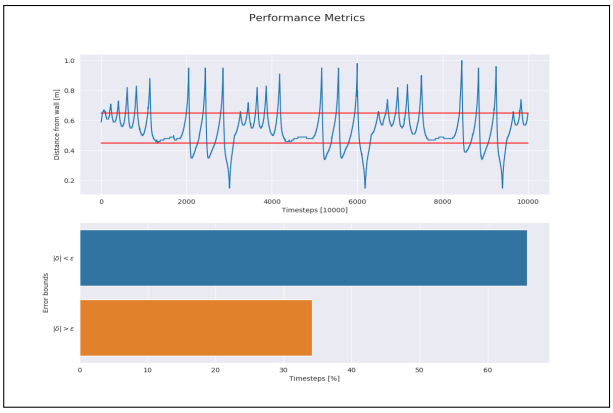

Figure 78. Nominal Policy without online update in custom world(1) (η = 1.0; sample 8)

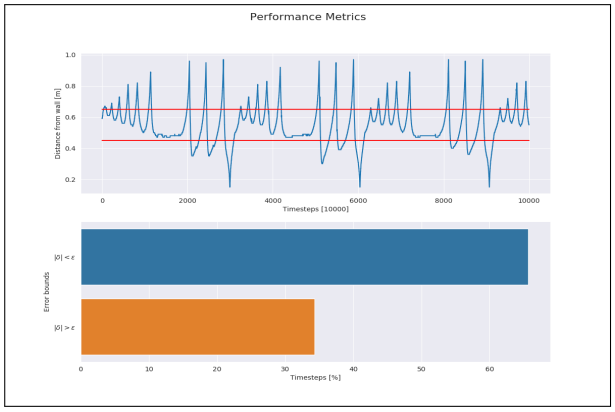

Figure 76. Nominal Policy without online update in custom world(1) (η = 1.0; sample 6)

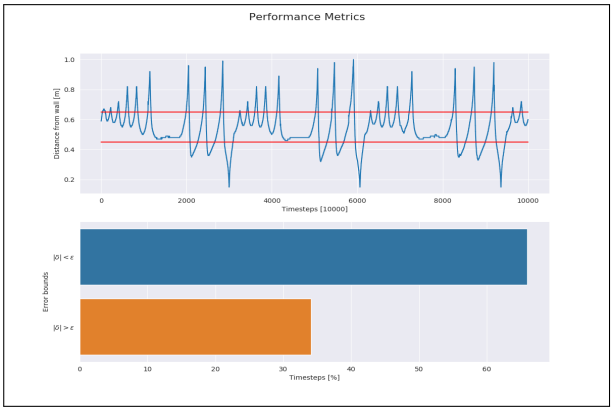

Figure 79. Nominal Policy without online update in custom world(1) (η = 1.0; sample 9)

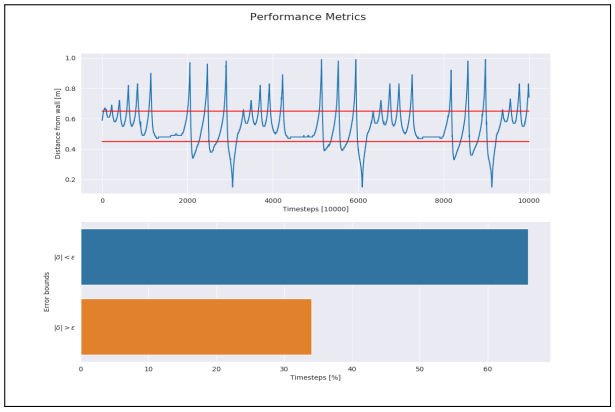

Figure 77. Nominal Policy without online update in custom world(1) (η = 1.0; sample 7)

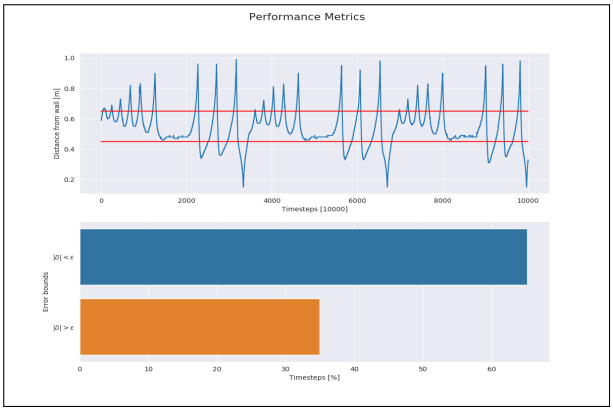

Figure 80. Nominal Policy without online update in custom world(1) (η = 1.0; sample 10)

B. Appendix

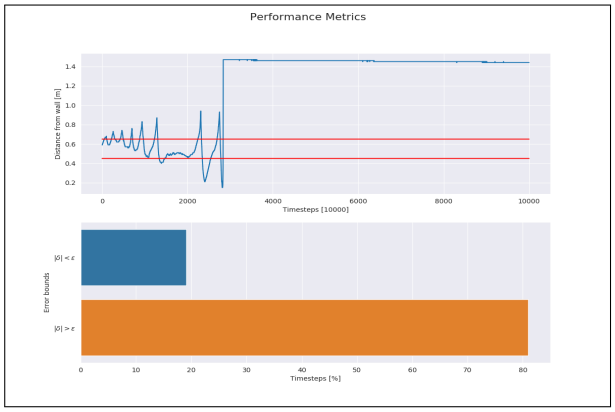

Figure 81. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 1)

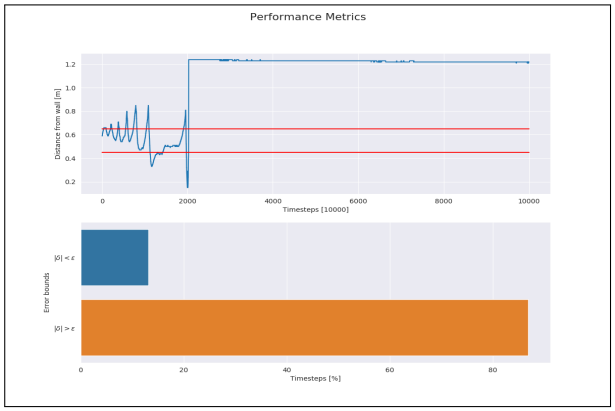

Figure 82. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 2)

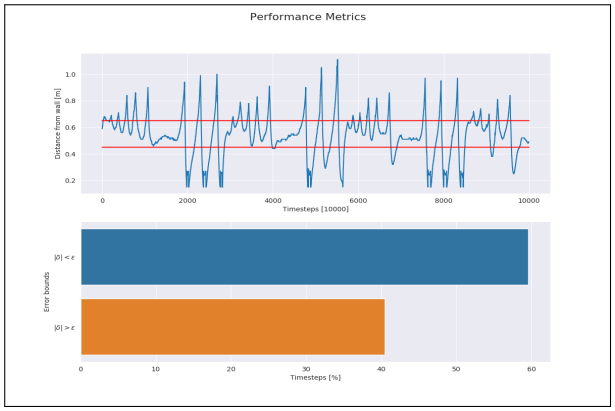

Figure 83. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 3)

Figure 84. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 4)

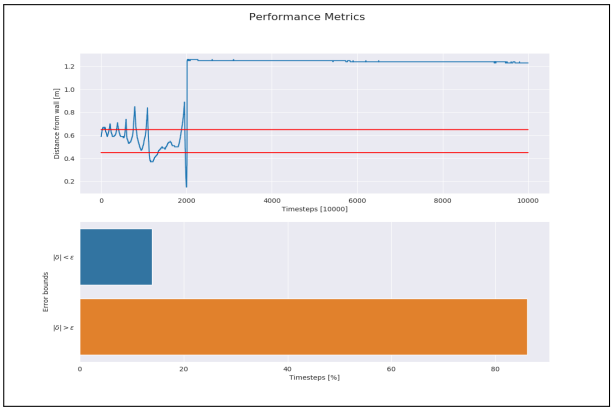

Figure 85. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 5)

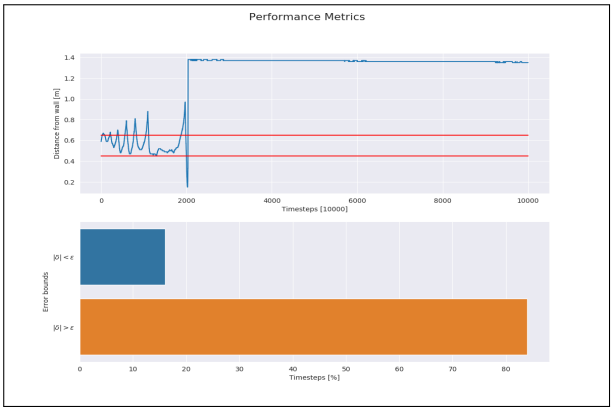

Figure 86. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 6)

Figure 87. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 7)

Figure 90. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 10)

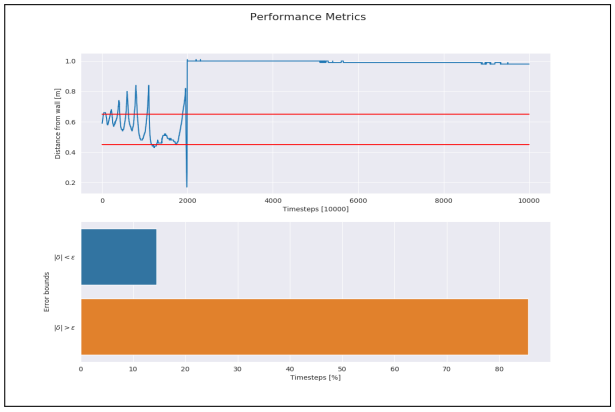

Figure 88. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 8)

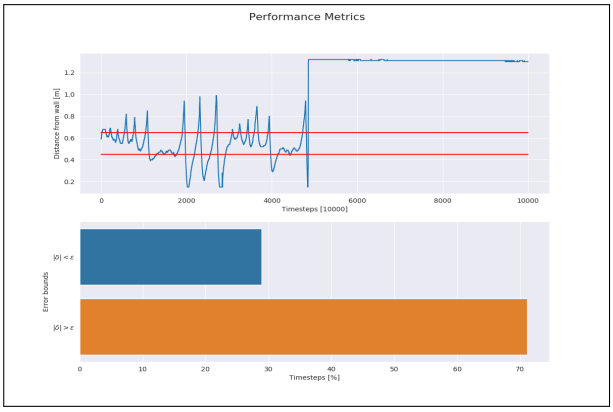

Figure 91. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 1)

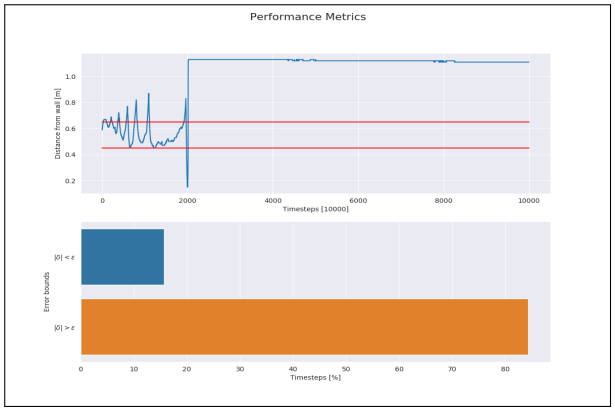

Figure 89. Nominal Policy with online update in custom world(1) ($\eta = 0.5$; sample 9)

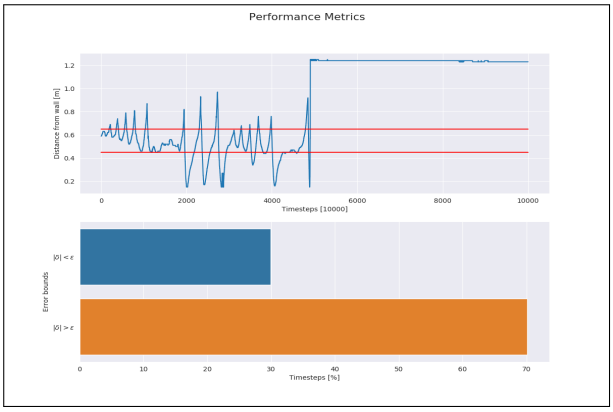

Figure 92. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 2)

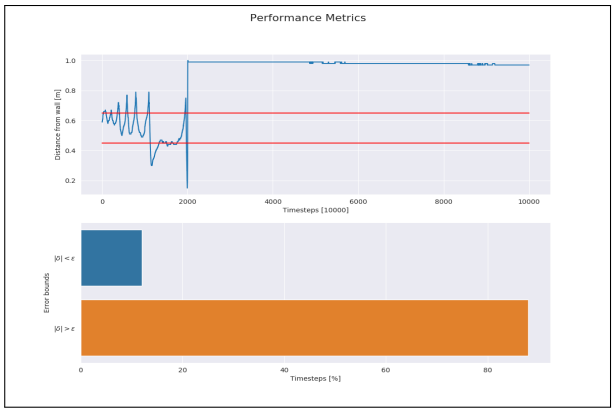

Figure 93. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 3)

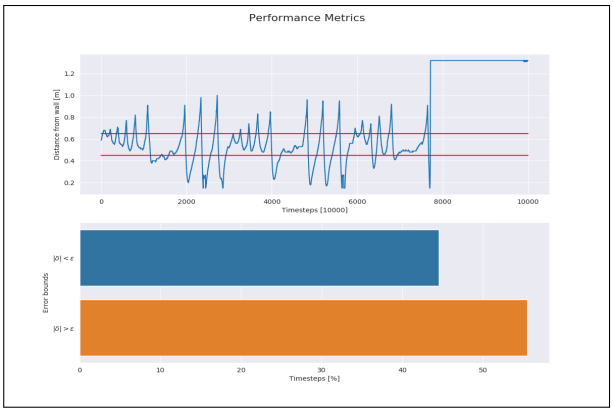

Figure 96. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 6)

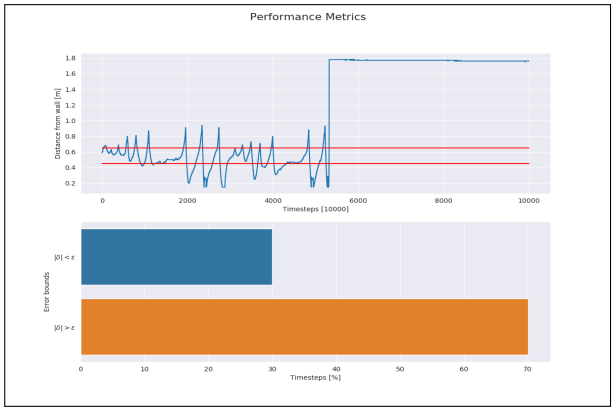

Figure 94. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 4)

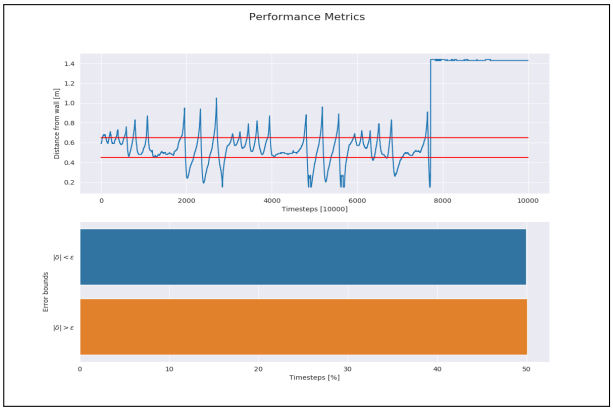

Figure 97. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 7)

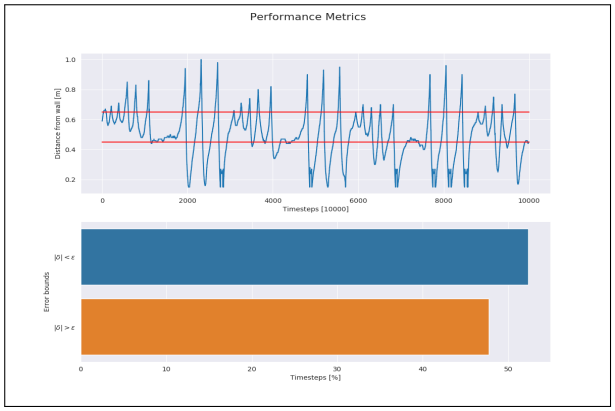

Figure 95. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 5)

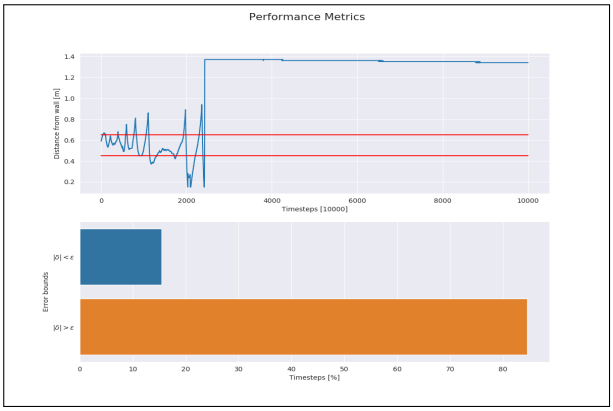

Figure 98. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 8)

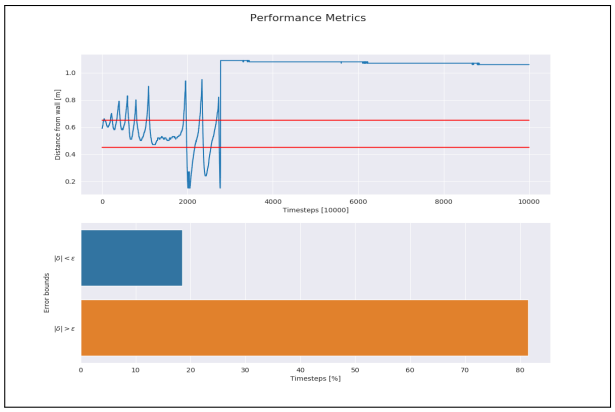

Figure 99. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 9)

Figure 102. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 2)

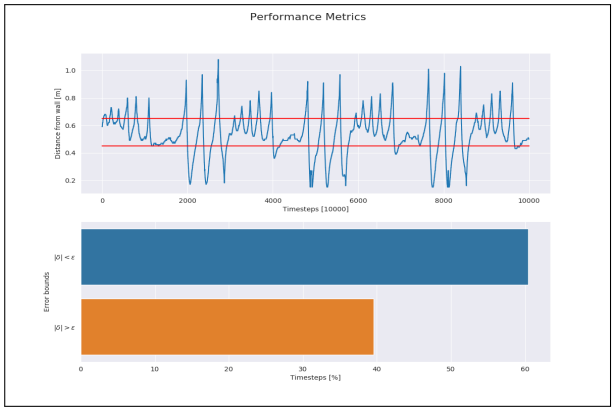

Figure 100. Nominal Policy with online update in custom world(1) ($\eta = 0.6$; sample 10)

Figure 103. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 3)

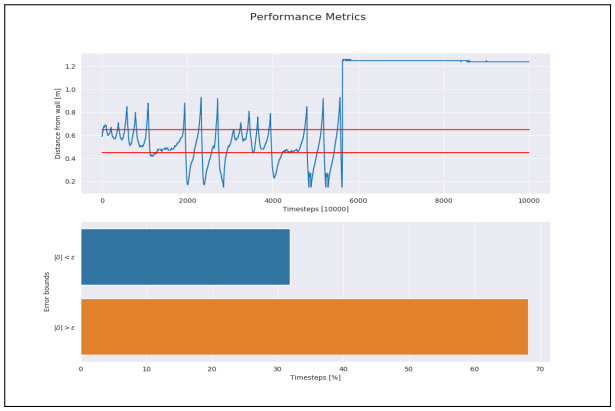

Figure 101. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 1)

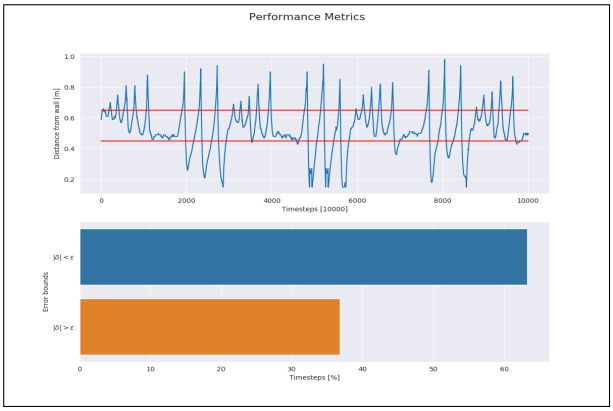

Figure 104. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 4)

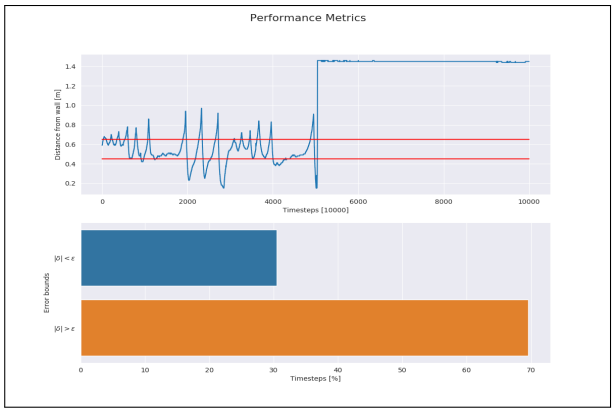

Figure 105. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 5)

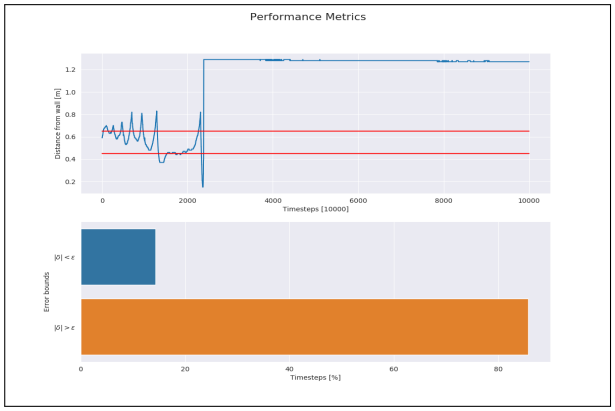

Figure 106. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 6)

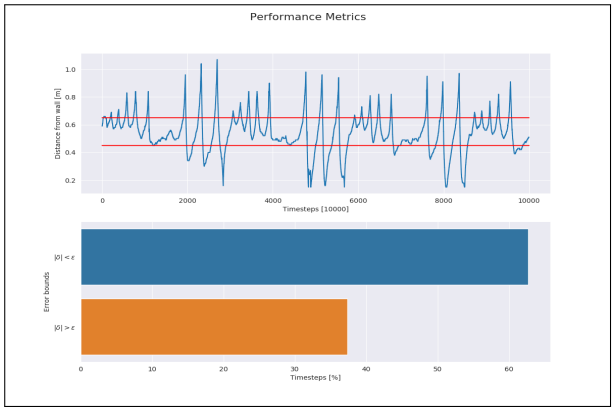

Figure 107. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 7)

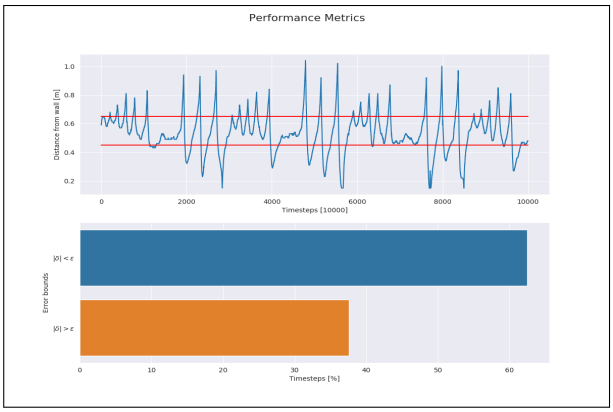

Figure 108. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 8)

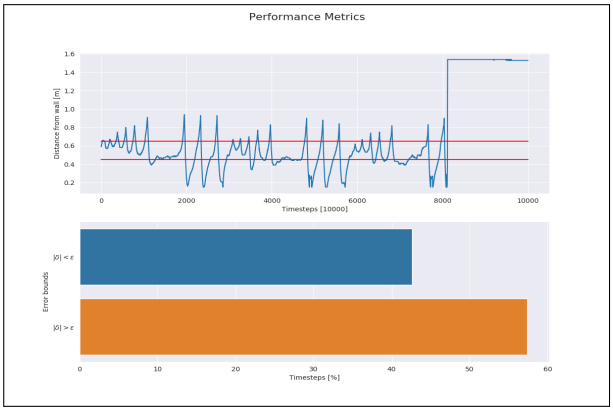

Figure 109. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 9)

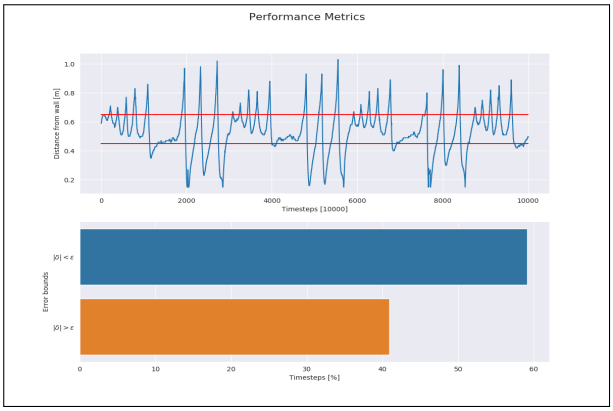

Figure 110. Nominal Policy with online update in custom world(1) ($\eta = 0.7$; sample 10)

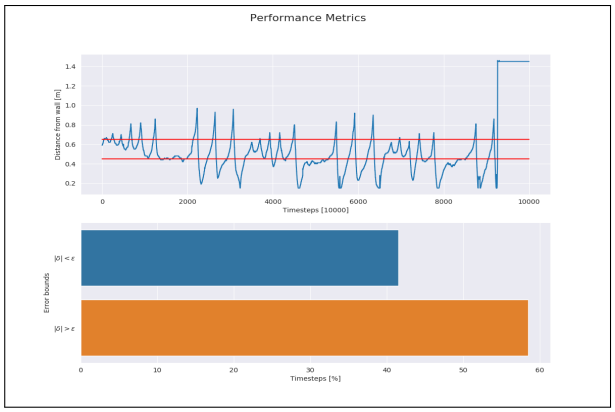

Figure 111. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 1)

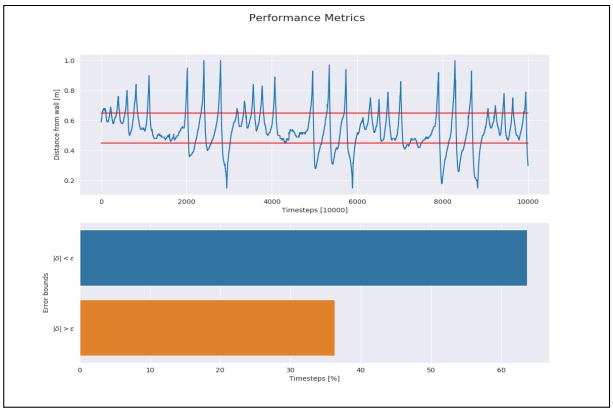

Figure 114. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 4)

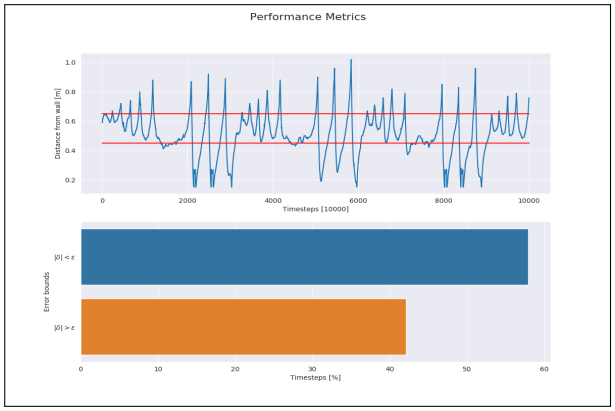

Figure 112. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 2)

Figure 115. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 5)

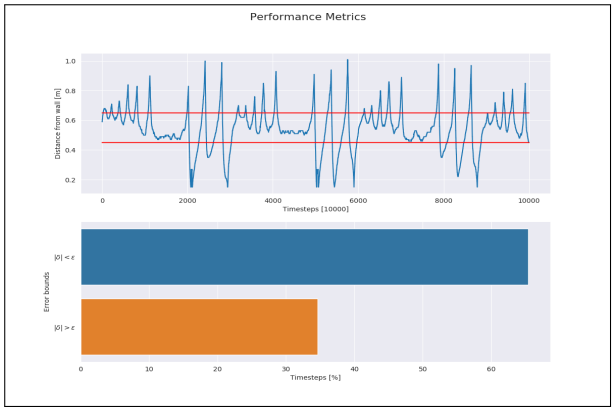

Figure 113. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 3)

Figure 116. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 6)

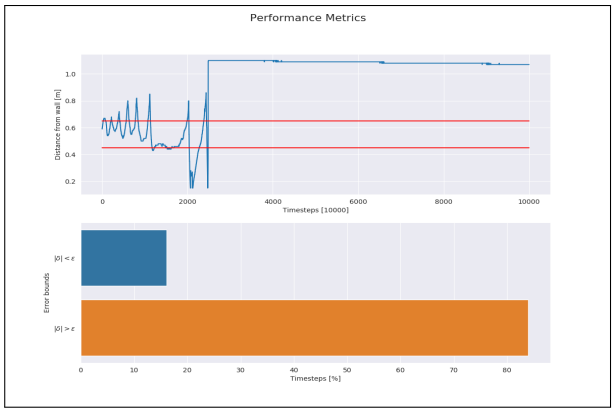

Figure 117. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 7)

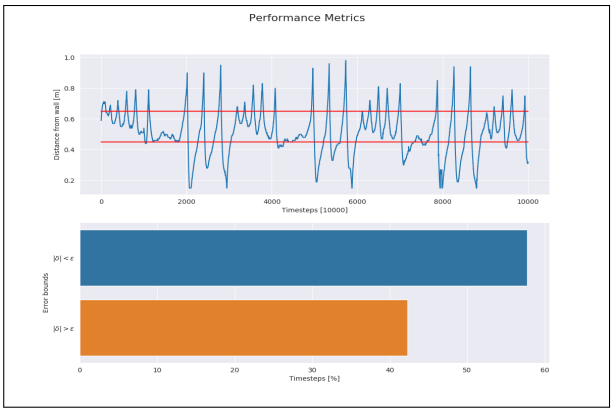

Figure 120. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 10)

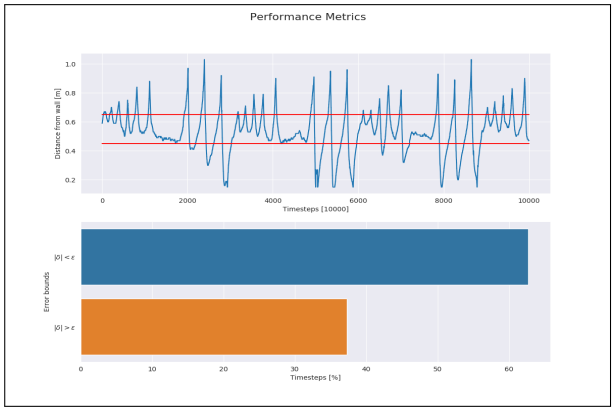

Figure 118. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 8)

Figure 121. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 1)

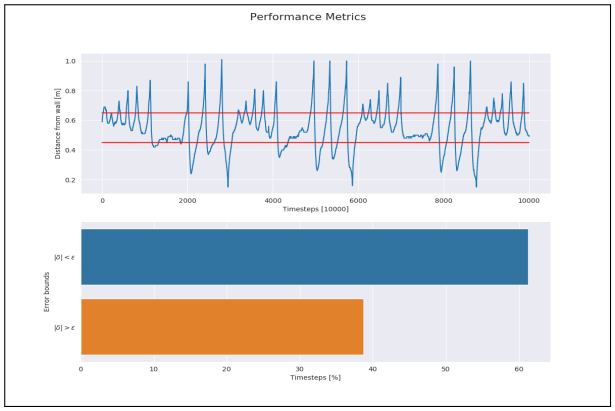

Figure 119. Nominal Policy with online update in custom world(1) ($\eta = 0.8$; sample 9)

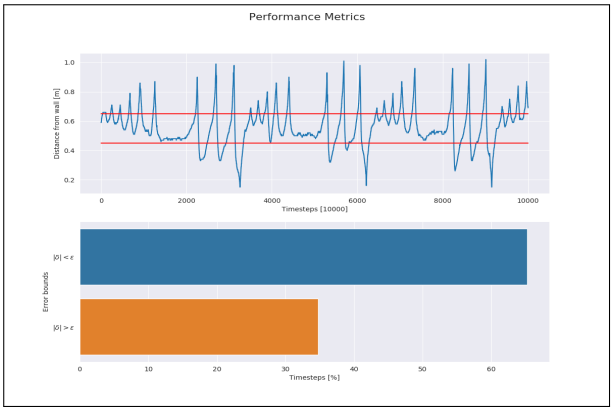

Figure 122. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 2)

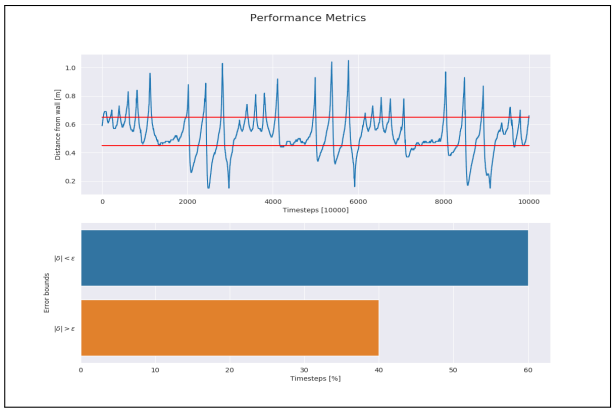

Figure 123. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 3)

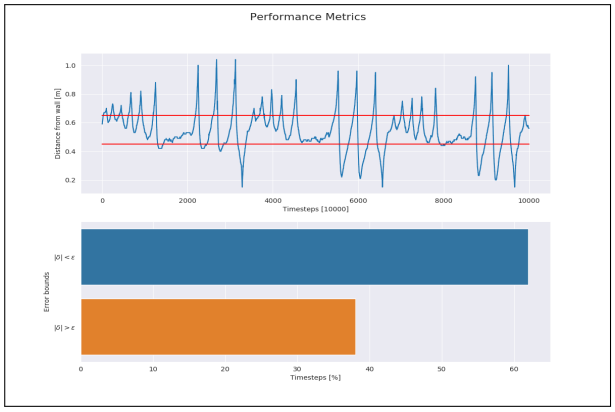

Figure 124. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 4)

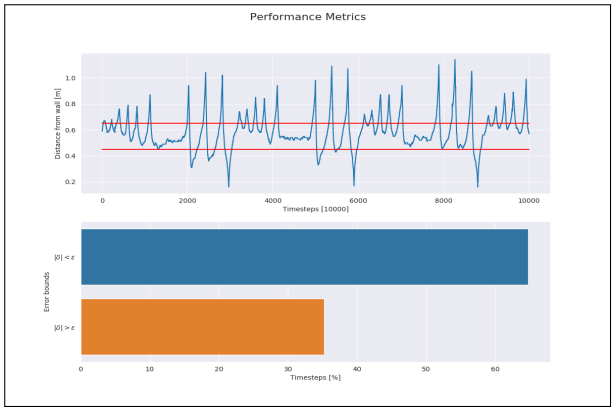

Figure 125. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 5)

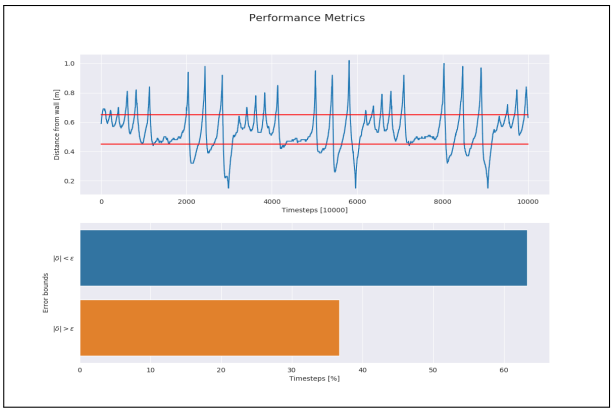

Figure 126. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 6)

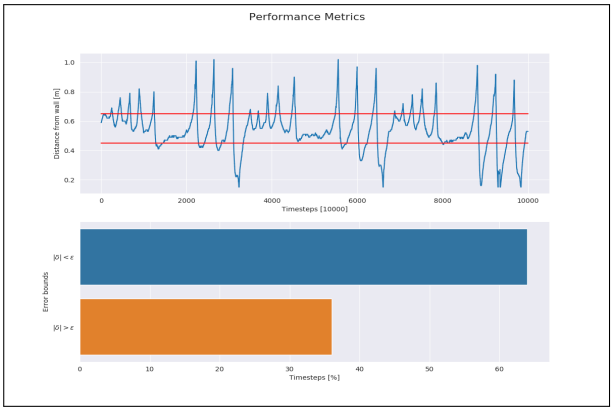

Figure 127. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 7)

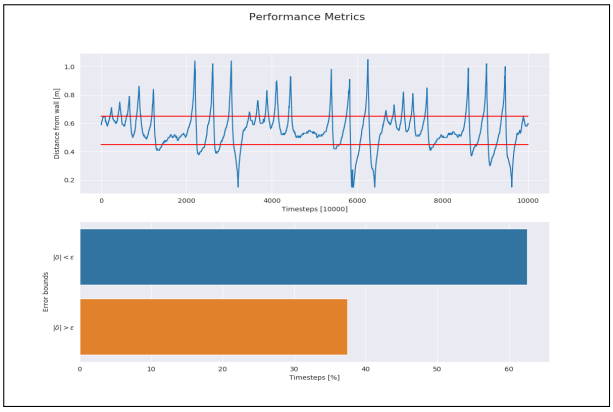

Figure 128. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 8)

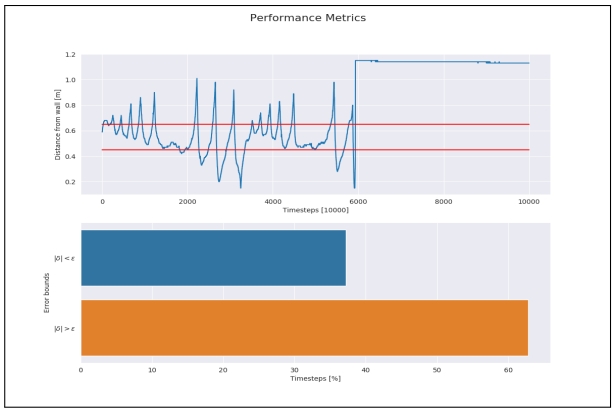

Figure 129. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 9)

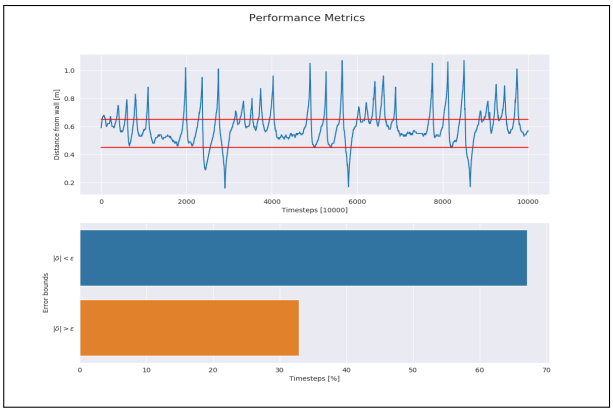

Figure 132. Nominal Policy with online update in custom world(1) ($\eta = 1.0$; sample 2)

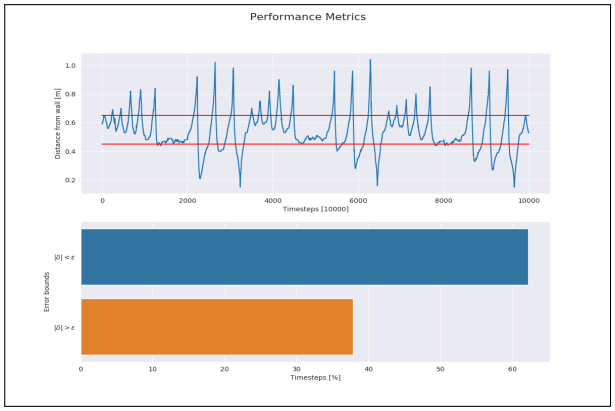

Figure 130. Nominal Policy with online update in custom world(1) ($\eta = 0.9$; sample 10)

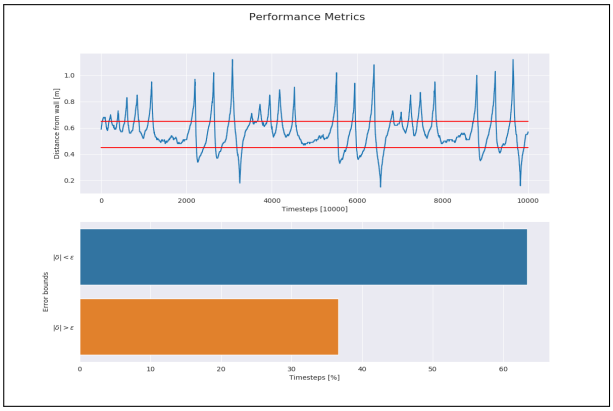

Figure 133. Nominal Policy with online update in custom world(1) ($\eta = 1.0$; sample 3)

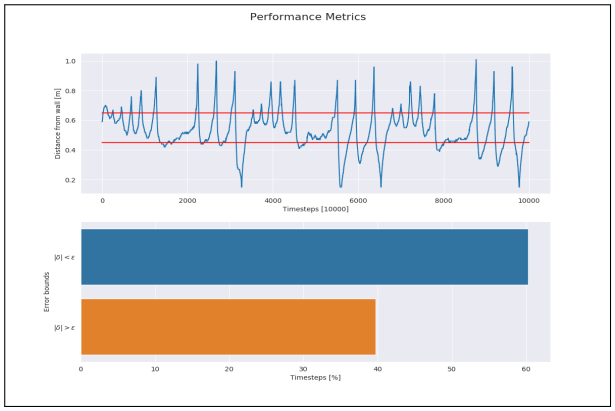

Figure 131. Nominal Policy with online update in custom world(1) ($\eta = 1.0$; sample 1)

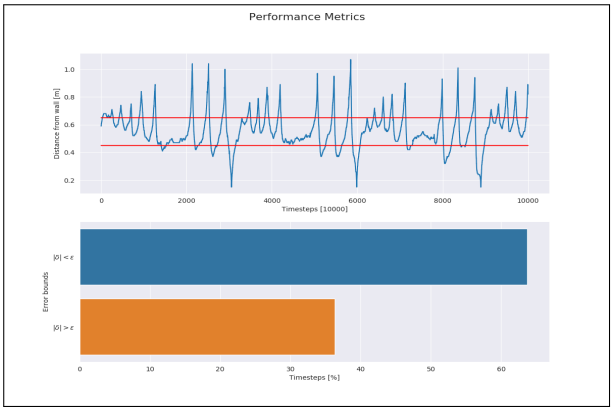

Figure 134. Nominal Policy with online update in custom world(1) ($\eta = 1.0$; sample 4)

Figure 135. Nominal Policy with online update in custom world(1) (η = 1.0; sample 5)

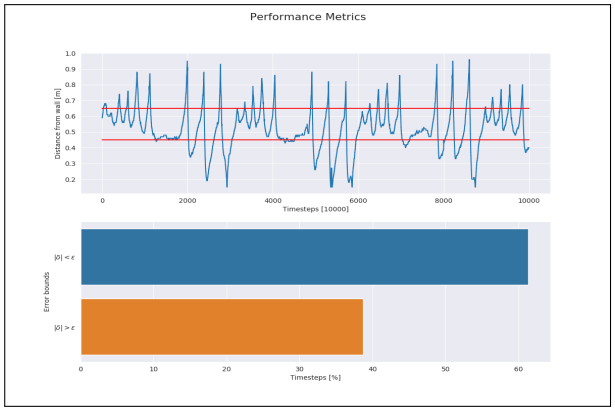

Figure 136. Nominal Policy with online update in custom world(1) (η = 1.0; sample 6)

Figure 137. Nominal Policy with online update in custom world(1) (η = 1.0; sample 7)

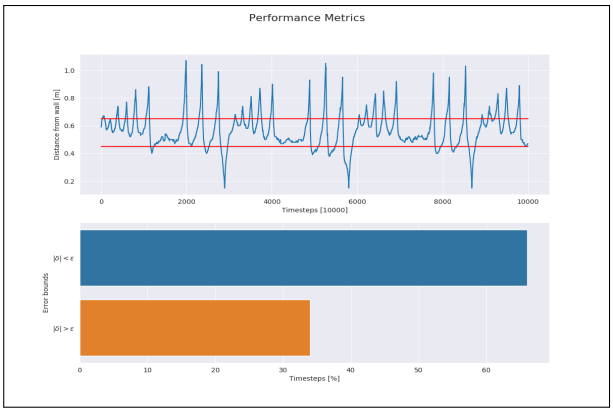

Figure 138. Nominal Policy with online update in custom world(1) (η = 1.0; sample 8)

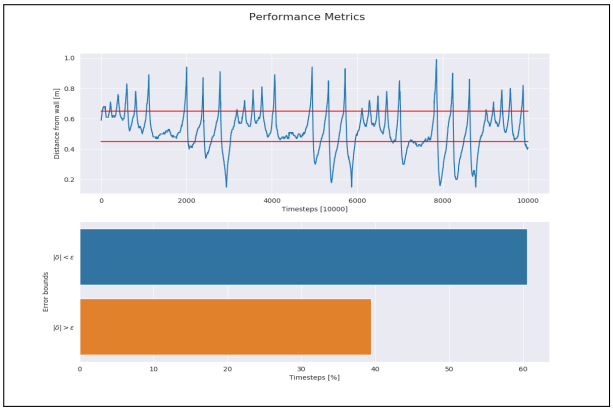

Figure 139. Nominal Policy with online update in custom world(1) (η = 1.0; sample 9)

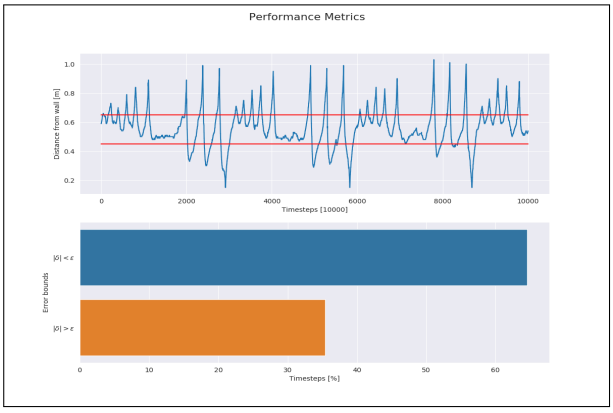

Figure 140. Nominal Policy with online update in custom world(1) (η = 1.0; sample 10)