A Comparison of Reactive Obstacle Avoidance Algorithms for a Ground Vehicle

Simon Hermansson Marcus Gandal Kevin Bärudde Hugo Axandersson

Daniel Covarrubias Gillin

Abstract

This project evaluates if the methods DDPG *and* MPC *can be used for the problem of reactive collision avoidance for unmanned ground vehicles. These methods are compared against a baseline method,* DWA*, which is already implemented in* ROS*. This paper concludes that while* DDPG *and* MPC *were not able to reach the performance of* DWA*, both methods can still be used for reactive collision avoidance. It is also noted that future improvement in the implementation of both methods is possible.*

Git: `https://gitlab.liu.se/tdde19-2022-2`

1. Introduction

The goal of this project was to investigate reactive obstacle avoidance algorithms for an Unmanned Ground Vehicle (UGV); specifically, the Clearpath Husky, which can be seen in figure 1. The Husky is a large robot, and thus it must have the ability to safely navigate complex terrain without damaging expensive equipment or humans. To aid the Husky in understanding the world, it is equipped with a LiDAR, which generates 3D point clouds describing the environment.

Figure 1: The Clearpath Husky.

1.1. Problem

Two methods are intended for the implementation, Model-Predictive Control (MPC) and Deep Deterministic Policy Gradient (DDPG). These methods should both allow for collision-free navigation. The methods should, given a goal coordinate and the current sensor readings, compute trajectories for the Husky until it reaches the goal state. The performances of the methods are compared against each other, and also against a baseline method, the Dynamic Window Approach (DWA), which is already implemented in the Robot Operating System (ROS). A program, Gazebo, will be used to simulate an environment for the Husky.

1.2. Assumptions

This paper does not explore the problem of building maps of the environment, therefore, it is assumed that the ground vehicle is always aware of its position in the world; as if it was equipped with a GPS. The ground is assumed to be flat and the obstacles are assumed to be static.

2. Theory

The theory behind the methods that will be used is presented in this chapter.

2.1. Dynamic Window Approach

The Dynamic Window Approach (DWA) is a common reactive collision avoidance method for autonomous ground vehicles; the method was presented by Fox et al. [2] in 1997. The *dynamic window* is the search space of velocities reachable within a short specified time interval. A combination of translational and rotational velocities is chosen by maximizing an objective function that can be seen in equation 1. In equation 1, h is a heading measure that is maximized when the robot moves directly towards the target, d is the distance to the nearest obstacle, and v is the forward velocity of the robot. v and w are translational and rotational velocities respectively, and σ smoothes the weighted sum which results in more side-clearance from the obstacle. Only admissible velocities, yielding trajectories in which the robot can stop safely before it reaches the closest obstacle are considered.

Maximization is done by first discretizing the search space down to the admissible linear and angular velocities, and then selecting the optimal velocities that maximize the objective function.

$$G(v, w) = \sigma(\alpha \cdot \mathrm{h}(v, w) + \beta \cdot \mathrm{d}(v, w) + \gamma \cdot \mathrm{v}(v, w) \quad (1)$$

DWA is already implemented within ROS, and it can therefore be used as a baseline when comparing against the other two methods.

2.2. Model-Predictive Control

Model-Predictive Control (MPC) is a control engineering method that is iterative, where a solution is generated each time step by optimizing a finite future of predictions [7]. MPC contains two parts, the controller and the plant, see figure 2. The controller takes a reference for the target state and feedback from the world and then returns manipulation variables that control the system. Inside the controller, there is a plant module and an optimizer. The plant module predicts future timesteps, while the optimizer selects a set of parameters to test and uses the result from the plant module to select the preferred manipulated variables based on the system constraints and a cost function. The plant part of the system is the actual robot. The system uses the modification variables to steer, and sensors are used to get feedback from the robot to the controller.

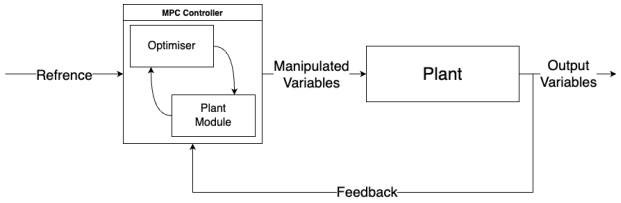


Figure 2: MPC.

Bayesian Policy Optimization Model-Predictive Control (BPO-MPC) is a real-time method for collision avoidance proposed by Andersson et al. [1]. They use an MPC controller of the form in equation 2, where $\mathbf{x} \in \mathbb{R}^n$ is a state vector, and $\mathbf{u} \in \mathbb{R}^m$ is a control vector. Equation 3 defines the transition dynamics, where θ_{dyn} is a simple linear model learned from data using maximum likelihood. Equation 4 defines the convex task constraints, and equation 5 defines the geometric obstacle constraints, which are concave. $m(\theta, \mathbf{x}_t)$ is a parametric safety margin given by a policy parameter vector θ that is learned using policy search. $\mathbf{p}_{r,t}$ is a subspace of $\mathbf{x}_t$, and $\mathbf{p}_{o,t}$ are constants in the optimization. Equation 6 defines penalty terms for violation of elastic constraints.

$$\underset{\mathbf{u}_0 \dots \mathbf{u}_{T-1}, \mathbf{x}_1 \dots \mathbf{x}_T, \epsilon_1 \dots T}{\arg\min} \mathbb{E}\left[\sum_{t=1}^{T} c(\mathbf{x}_t, \mathbf{u}_{t-1})\right] \quad (2)$$

subject to

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}; \theta_{dyn}), \quad (3)$$

$$\mathbf{g}_{task}(\mathbf{x}_t, \mathbf{u}_{t-1}) \geq -w_{task}^{-1} \epsilon_{t,task}, \quad (4)$$

$$\mathbb{E}\left[dist(\mathbf{p}_{r,t}, \mathbf{p}_{o,t})\right] - m(\theta, \mathbf{x}_t) \geq w_{obst}^{-1} \epsilon_{t,obst}, \quad (5)$$

$$\epsilon_t \geq 0, \text{ where } t = 1, ..., T \quad (6)$$

Andersson et al. have done work with BPO-MPC together with a drone. This research will be adapted to the ground robot, the Husky, which is used in this paper.

2.3. Reinforcement Learning with DDPG

Lillicrap et al. [6] present an actor-critic deep reinforcement learning method called *Deep Deterministic Policy Gradient* (DDPG), designed for high-dimensional, continuous action spaces. In their paper, they present the results from applying this algorithm on over 20 simulated physics tasks with varying complexity, including a self-driving car. The algorithm can be seen in algorithm 1. For details, we refer to the original authors' paper [6]; however, the core components are explained briefly in this section.

2.3.1 Replay Buffer

The replay buffer stores previous transitions that the agent has taken which are used as training data for the networks. The idea of the replay buffer was proposed in a paper by Mnih et al. [8]. A transition is of the following form: (s_t, a_t, r_t, s_{t+1}), where s_t is the initial state at time t, and a_t is the action taken from this state to the new state s_{t+1}, resulting in the reward r_t.

2.3.2 The Four Networks

The actor network is trained to predict the optimal action to take given a state, and the critic is trained to predict how good the action taken was. It is analogous to the classical reinforcement learning paradigm in the way that the actor represents the policy and the critic represents the value function [3]. The policy is deterministic in the sense that the action will always be the same given a state [6]. The actor and critic are neural networks trained together to learn the parameters of the optimal policy and value function.
During each time step, a mini-batch consisting of a user-defined number of transitions is sampled from the replay buffer to perform one training iteration of the networks (forward and backward propagation). Details can be found in algorithm 1. The critic network is updated using the temporal difference between the target value y_i and the current

value $Q(s_i, a_i|\theta^Q)$. The temporal difference is minimized using the Mean-Square-Error (MSE) loss function. Furthermore, the user-defined discount factor γ is used to regulate the importance of future rewards. The actor is updated using the policy gradient. The actor loss is defined by the gradient of the critic value with respect to the parameters of the actor network θ^μ. Finally, the target networks are updated at a slower rate compared to the ordinary networks, which can be set by the user-defined parameter τ, to stabilize the learning.

2.3.3 Exploration Noise

The exploration noise, denoted by $\mathcal{N}$, is used to encourage the exploration of new actions and states. In classical reinforcement learning algorithms, the exploration-exploitation dilemma is defined as the balance between taking the best currently known action in a given state or to explore new paths that can potentially result in higher rewards. The epsilon-greedy policy first proposed by Watkins [11] is a popular strategy to address this. When working with continuous action spaces, one way of implementing exploration is to sample some random noise from a random process and add that to the predicted action by the actor [6]. The proposed random process by the authors of DDPG is the Ornstein-Uhlenbeck process [10].

2.3.4 Related Work

Tai et al. [9] present a paper in which they successfully implement DDPG to solve the problem of continuous control of a TurtleBot3 for map-less navigation. In their work, a learning-based map-less motion planner is implemented that takes 10-dimensional range findings and the target position as inputs and returns the continuous steering commands for the linear and angular velocities; an illustration of this concept can be seen in figure 3. Once trained, the planner was applied to new, previously unseen environments. The actor and critic network structures proposed in their work can be seen in figure 4.

Following the work of Tai et al., Jesus et al. [5] present a paper in which they demonstrate the effectiveness of mobile robot navigation with DDPG in simulated Gazebo environments using a TurtleBot3. Their robot manages to reach its goal in multiple environments, whilst successfully avoiding both static and dynamic objects along its path. In their work, they apply the same design concept as proposed by Tai et al., shown in figure 3, and the same network structures for the actor and critic networks, shown in figure 4.

3. Methodology

The implementation of the suggested methods is described in this chapter.

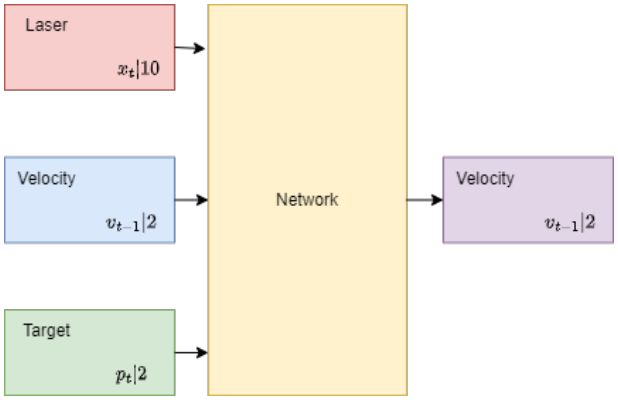


Figure 3: System overview.

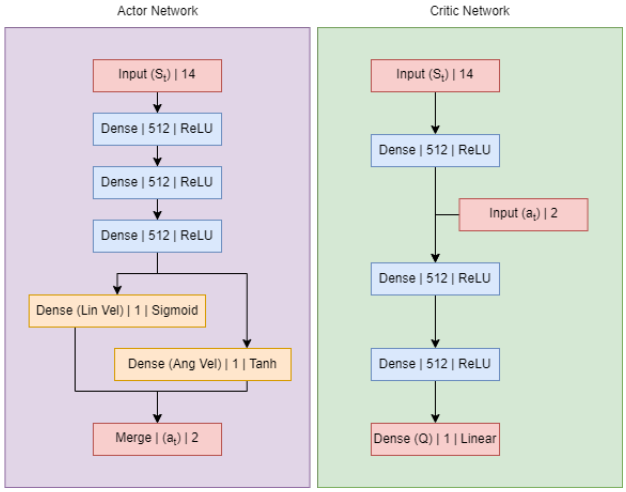


Figure 4: Actor-Critic networks.

Algorithm 1 The DDPG algorithm.

Random initialize critic $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ networks with weights θ^Q and θ^μ
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss:
    $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:
    $\nabla_{\theta^\mu} J \approx \nabla_{\theta^\mu} Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i|\theta^\mu)}$
    Update the target networks:
    $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
    $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
  end for
end for

3.1. DWA

For DWA, the *husky_navigation* ROS package was used to gather baseline data. For input data, the LMS1XX LiDAR was used instead of the Velodyne VLP-16 LiDAR that was used for MPC, since it was already preconfigured.

3.2. Model-Predictive Control

The MPC planner was implemented with the help of the ACADO toolkit [4], which among many functionalities, offers support for generating MPC solvers. To be able to generate an MPC solver, the characteristics of the UGV had to be defined. DIFFERENTIALSTATE variables were used to define values such as the UGV position, velocity, and acceleration. How these values were connected to each other was then specified by, for example, stating that the differential of the heading was equal to the angular velocity.
To include obstacle avoidance into the MPC problem definition, the VLP-16 LiDAR sensor readings for the UGV were flattened into a 2D plane using the *pointcloud_to_laserscan* ROS package, and then divided into 10 sectors. Within each of these sectors, the position of the closest object was calculated and later used in the MPC problem definition. ONLINEDATA and INTERMEDIATESTATE variables were used to save the positions and proximities to the obstacles respectively. These variables can be continually updated during execution.

3.2.1 Cost Function

The MPC problem definition can be divided into two parts, the cost function, and the constraints. The cost function contains a linear combination of the following values:

1. The UGV position relative to the target, to encourage it to move towards the target.
2. The angle of the UGV relative to the target angle towards the target position, to encourage it to turn towards the target.
3. The UGV forward velocity and acceleration, to discourage it from moving too fast.
4. The UGV angular velocity and acceleration, to discourage it from turning too fast.
5. A value for the general proximity to obstacles around the UGV (GP as defined in equation 7), to discourage it from moving too close to obstacles.

Most of the weights of the cost function were set to 1; the exceptions being the weights for the position and the general proximity, which were set to 10 and 0.1 respectively. The distance from a variable to its target state is not normalized. Therefore, a variable such as the positional variable, which can be quite far away from its target value, is naturally already weighed more than the acceleration variables, which take on quite small values. The cost function is finally minimized using least squares minimization.
The value for the general proximity used in the cost function is defined in equation 7 where m is a set margin, currently set at 1 meter, and d_i is the distance to the closest object within sector i.

$$GP = \sum_{i=1}^{10} e^{2m-d_i} \tag{7}$$

The reasoning for using an exponential for the definition of the GP was that the cost function should care very little when obstacles are far away and care very much when they are within 2 meters.

3.2.2 Constraint Function

Along with the cost function, constraints were also defined for the MPC problem. The constraints were primarily used to describe the limitations of the UGV movement, such as velocity and acceleration constraints; the exception being the constraint in equation 8 which was used to keep the UGV a margins distance away from obstacles.

$$p_i \geq m \; \forall i \in [1, 10] \tag{8}$$

The cost function and the constraints, along with sensor data, were then used by ACADO to generate a control signal that the UGV could act upon.

3.3. DDPG

An overview of the system can be seen in figure 3. The system receives as input a state consisting of 15 values. It included 11 evenly distributed laser readings from the LMS1XX LiDAR ranging from -90 degrees to +90 degrees relative to the robot's heading. The maximum range considered was 10 meters. The state also included the linear and angular velocities of the robot and the distance and heading relative to the target. The distance to the target was normalized by the diagonal length of the environment and the heading relative to the target was normalized by 2π.
The network architectures used can be seen in figure 4. The actor network received a state as input and returned as output the new linear and angular velocities of the robot constrained to the intervals [0.0, 0.7] m/s and [-0.5, 0.5] rad/s respectively. The critic also received the state as input and the action predicted by the actor network. As exploration noise, values sampled from a uniform distribution in the intervals [0.0, 1.0] for linear velocity and [-0.5, 0.5] for angular velocity were used.

3.3.1 Training

The robot was trained in the Gazebo simulation environment in a total of two different environments. The first environment was a straight 10-meter corridor with the goal placed at the end, see figure 5. This environment included random spawns for the robot at the beginning of each episode. The second environment was a 20x20-meter map filled with various obstacles, see figure 6. Both random spawns and random goal positions were used when training the robot in this environment.

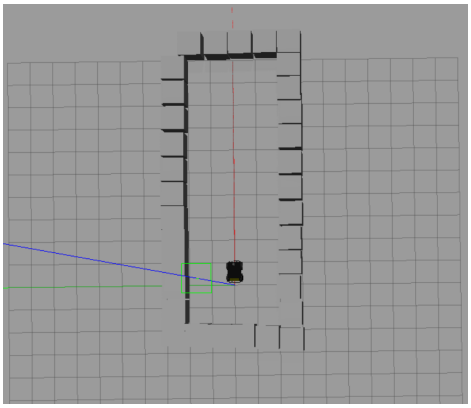


Figure 5: Corridor training map.

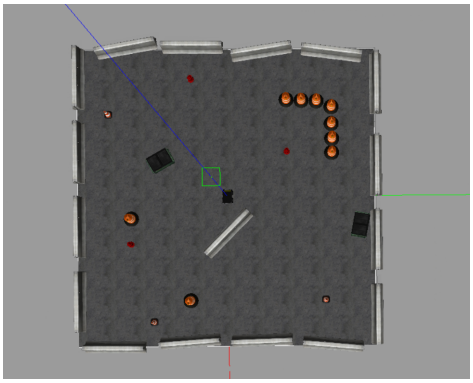


Figure 6: Playpen training map.

The first environment was only used for pre-training the networks. The pre-training was implemented to address the problem of the robot learning to spin around in circles indefinitely. For the pre-training, 5000 transitions from the first environment were collected and saved in the replay buffer using random actions sampled from a uniform distribution. The networks were then trained for 1000 iterations on this data. After the pre-training phase, the robot was moved to the advanced environment where it was further trained for 700 episodes. During this training phase, the proposed DDPG algorithm was used, see algorithm 1, the only difference being that the replay buffer was not empty at the start of the algorithm and the network weights had already been pre-trained. Each episode lasted a maximum of 90 actions and each action was taken for exactly 1 second.

$$R = \begin{cases} 120 & \text{when reaching the goal} \\ -100 & \text{when colliding with an obstacle} \\ 500 * (S_{t-1} - S_t) & \text{otherwise} \end{cases}$$

Figure 7: DDPG reward function.

The robot received rewards given in figure 7, where S_t is the distance of the robot from the goal at iteration t. The reward function motivated the robot to move towards the goal while avoiding obstacles along its path. Other reward values were tested but it was concluded that the ones presented yielded the best results. To determine if a collision had happened, the minimum range value was compared against a threshold of 0.5 meters. Similarly, the robot was considered to have reached the goal if it was within 2.0 meters of it. All the user-defined parameters involved for the reinforcement part are presented in table 1.

Parameter	Value
Actor learning rate	0.001
Critic learning rate	0.001
Target actor/critic learning rates	0.001
Discount factor	0.99

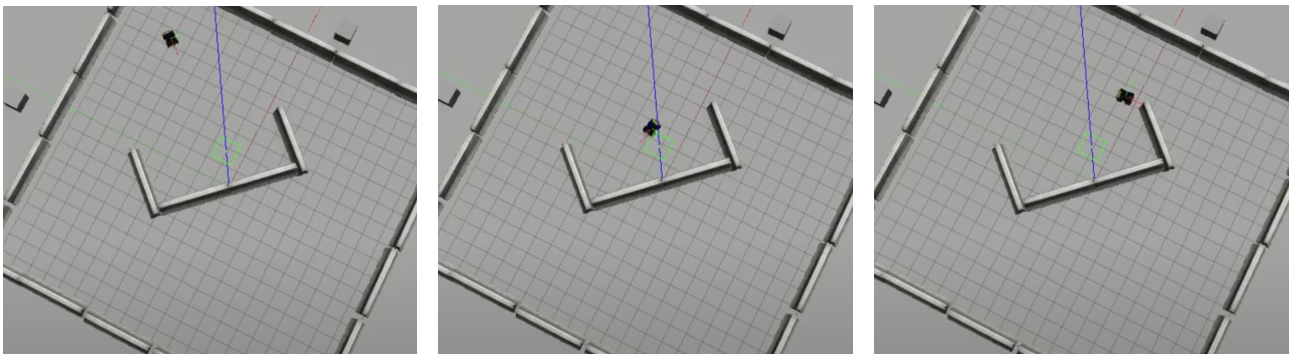
Table 1: DDPG parameters.

4. Results

The results of the methods are presented in this chapter.

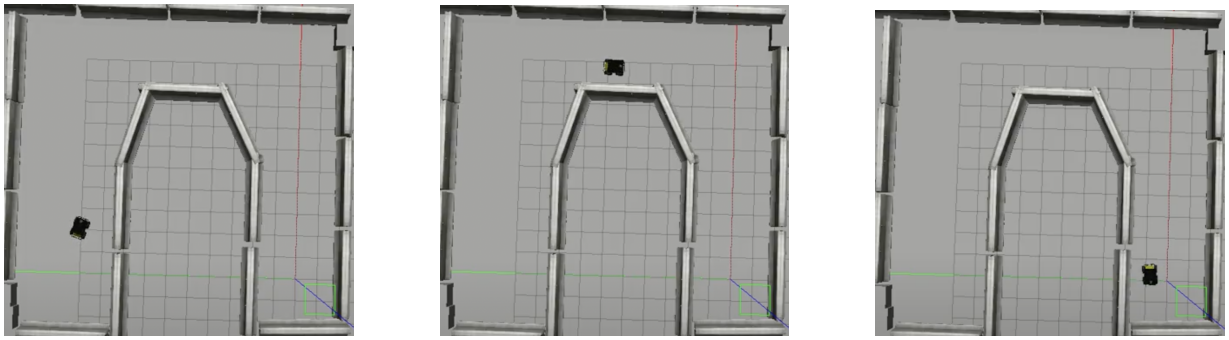
4.1. DWA

The baseline results from DWA are shown in figures 8, 9, and 10. The robot can avoid obstacles, even obstacles with concave shapes. The robot manages to find a path to the goal even when that path moves away from the goal. Some limitations of DWA are noted, where the method can make the UGV move very close to obstacles when it would result in being closer to the goal. This also results in the robot slowing down, examples of the UGV moving very close to obstacles are shown in figures 9b and 10c. The results also show that while it can move around the concave obstacles in figure 8c, the robot can easily get stuck when trying to move around the obstacle, due to not having a large enough safety margin.



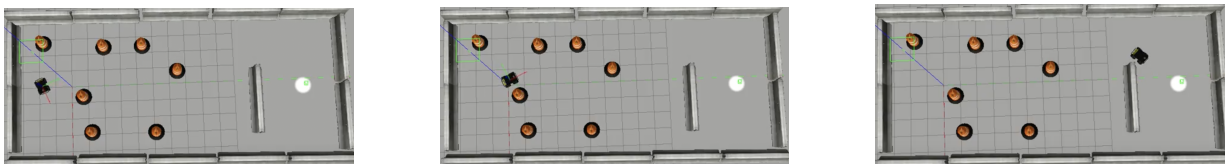
(a) Start position. (b) The Husky drives inside the obstacle. (c) Gets slightly stuck on the way out.

Figure 8: DWA: Concave obstacle map.



(a) Starting position. (b) The Husky hugs the corner. (c) The Husky reaches the goal.

Figure 9: DWA: U-shaped map.



(a) Starting position. (b) The Husky hugs the cone. (c) The Husky reaches the goal.

Figure 10: DWA: Multiple obstacles.

4.2. Model-Predictive Control

The results from the MPC system are displayed in figures 12, 11 and 13. The robot manages to reach the end of the multiple obstacles map in figure 11c while keeping a safe distance from the obstacles. The robot does not manage to escape the concave obstacle in figure 12b when moving inside it, and it also cannot find the path to the goal in the U-shaped map in figure 13.

4.3. DDPG

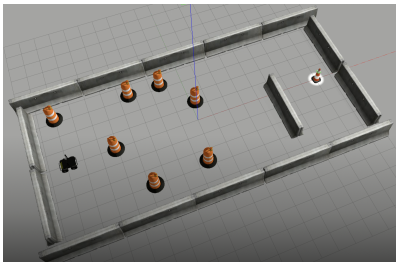
The results for DDPG are presented in figures 14, 15 and 16. The robot reaches the goal in the U-shaped map in figure 14c and the multiple obstacles map in figure 15c. However, the robot is not able to reach the goal in the concave map and gets stuck on the way to the target, as seen in figure 16b. It can be noted that the robot in many of the paths took very small margins by navigating very closely to the obstacles, see figures 15b and 15c.

5. Discussion

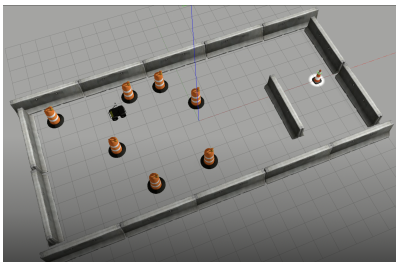
A discussion of the results obtained from the methods is presented in this chapter.

5.1. Model-Predictive Control

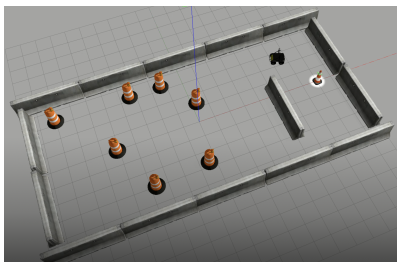
The results show that MPC can be used for collision avoidance. But it has its downsides compared to the DWA method. One downside is that it cannot explore the environment to find the goal as well. This could likely be fixed by making the robot value the goal state less, or by predicting more future timesteps. However, there are problems with predict-



(a) Starting position.

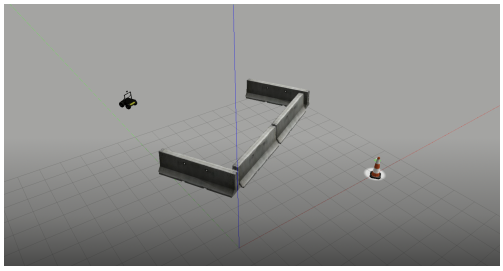


(b) The Husky avoided the cones.

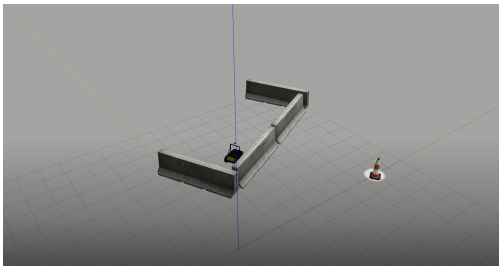


(c) Reaches the goal state.

Figure 11: MPC: Multiple obstacle map.



(a) Starting position.



(b) The Husky gets stuck.

Figure 12: MPC: Concave obstacle map.

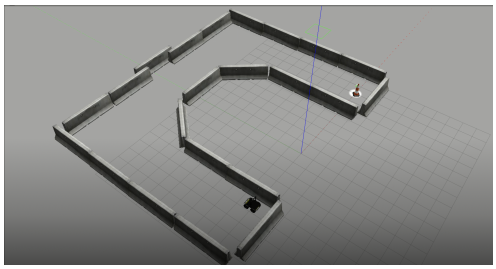


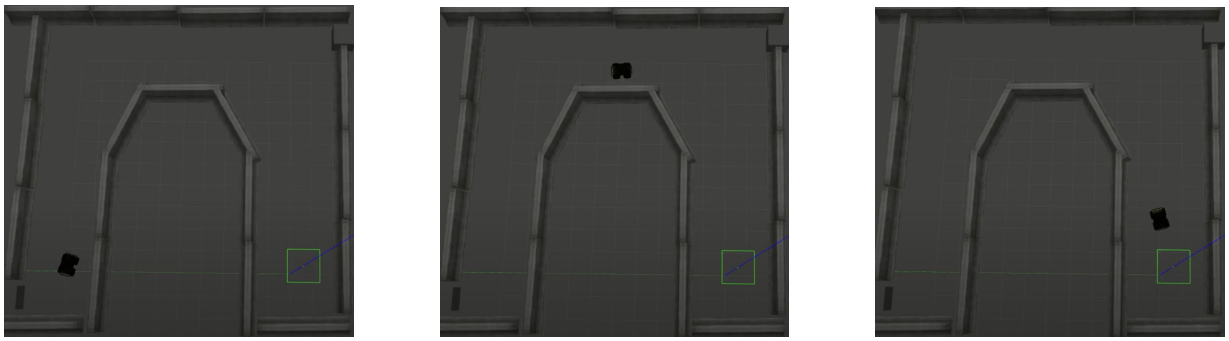
Figure 13: MPC: Stuck in U-shaped map.

ing additional timesteps. One problem is that it requires additional computational resources, and the 20 timesteps spaced 0.3 seconds apart that were used, were already pushing the maximum possible. Another problem is that predicting many timesteps requires a very good model of the system; a model that also takes into account any potential disturbances in the environment. This was not possible due to time limitations, and limitations of the simulation environment used, which could not recreate the real world accurately enough.

Much time was spent trying to optimize the parameters of the model, and it took a great deal of time to find reasonably stable parameters. Still, many fluctuations in the predictions between iterations can be noted when running the solver. Given more resources spent on accurately modeling disturbances, it is reasonable to think that the solver would be able to predict more stable solutions, which in turn would increase performance greatly.

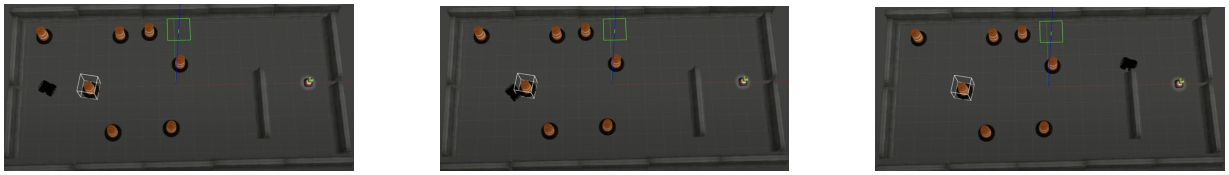
5.2. DDPG

The results indicate that although it is possible to apply the DDPG algorithm for robot navigation in simulated environments, it comes with its limitations. To start with, the robot was difficult to train. It took a lot of trial and error before the robot started to act as expected. The DDPG algorithm may be sensitive to divergence as the actor network frequently learned to output 0.0 in linear velocity and 1.0 in angular velocity, which meant that it kept driving around in circles. One possible explanation is that the networks overfitted on the few transitions in the replay buffer, which indicates that starting the DDPG algorithm with an empty memory buffer might impact the performance negatively. It could also have been that the reward function used did not correctly reflect the problem which we wanted to model and that the robot learned to accumulate positive rewards through a mean that was not intended. In this project, these problems were addressed by pre-training the networks as mentioned in sec-



(a) Starting position. (b) The Husky hugs the corner. (c) Reaches the goal state.

Figure 14: DDPG: U-shaped map.



(a) Starting position. (b) The Husky gets close to a cone. (c) Collides with the edge of the wall.

Figure 15: DDPG: Multiple obstacle map.

(a) Starting position. (b) The Husky gets stuck.

Figure 16: DDPG: Concave obstacle map.

tion 3.3.1.

For the environments where the robot did not successfully navigate to the target, as in figure 16b, some possible solutions can be conceived. One possible solution could be to tune the rewards so that a more appropriate balance between reaching the goal and avoiding obstacles could be achieved. Another way could be to let the robot train for longer.

Another point worth discussing is velocity. The robot often chooses to move at low velocity, and this is after considering its velocity constraints. It is likely easier to control the robot if it moves at a lower speed, which likely is why the agent learned to act this way. One possible solution to this could be to introduce a reward based on the number of time steps taken per episode, where fewer time steps taken per episode grants a higher reward. This type of reward could encourage the robot to move faster.

Finally, it is worth discussing the impact of the number of training episodes. During training, the moving average reward of the past 40 episodes was tracked. It was observed that the agent frequently became worse than it had been in the past; it seemed to forget what it has previously learned. Usually, it recovered from it and became better over a longer period of time. This speaks for the instability of the DDPG algorithm which the target networks were designed to combat, however, they do not seem to solve the problem entirely. Due to this instability and unpredictability, the training was concluded when the robot was deemed good enough, but it may have achieved better performance if given more training episodes.

6. Conclusions and Future Work

The experiments conducted in this project conclude that both DDPG and MPC can be used for reactive collision avoidance in a simulated environment, although not currently as well as DWA.

The DDPG method can adapt to new environments using knowledge learned from other environments, and it manages to both avoid obstacles and reach its destination. However, it does have its limitations, the DDPG proved to be unpredictable at times and difficult to train. DDPG did not achieve the same level of performance as DWA, however, its performance could likely be improved by extending the training time.

The MPC method, unlike DDPG, does not need to be trained. It seems to be more reactive, and fails more often when faced with large obstacles compared to both DWA and DDPG. However, the method still has potential, as the parameters reached were most likely not the optimal ones; better performance could likely be achieved by modeling the robot and the disturbances better.

References

[1] O. Andersson, M. Wzorek, P. Rudol, and P. Doherty. Model-predictive control with stochastic collision avoidance using bayesian policy optimization. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4597–4604. IEEE, 2016.

[2] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[3] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.

[4] B. Houska, H. Ferreau, and M. Diehl. ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[5] J. C. Jesus, J. A. Bottega, M. A. Cuadros, and D. F. Gamarra. Deep deterministic policy gradient for navigation of mobile robots in simulated environments. In *2019 19th International Conference on Advanced Robotics (ICAR)*, pages 362–367. IEEE, 2019.

[6] 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.

[7] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36(6):789–814, 2000.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.

[9] L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

[10] G. E. Uhlenbeck and L. S. Ornstein. On the theory of the brownian motion. *Phys. Rev.*, 36:823–841, Sep 1930.

[11] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge United Kingdom, 1989.