Real-Time Obstacle Avoidance for Mobile Robots using Deep Reinforcement Learning

SSY226, Group 1

Adam Burman[1] Ilya Kuangaliyev[1] Krister Mattsson[1] Gabriel Nyman[1] Arvid Petersén[1] Lukas Wisell[1]

***Abstract*—One part of creating effective, automated industries could be the implementation of Autonomous Transport Robots (ATRs) for transporting goods internally. This project concerns improving previous work regarding obstacle avoidance for such robots. The improvements mainly consist of implementing the Deep Reinforcement Learning (DRL) algorithms Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3) in an existing framework. There, the output from the DRL models is used as a reference to a model predictive controller, which controls the ATRs actions. Evaluations of the new control scheme show promising results that outperform previous methods. Several areas can be improved, chief among which is the extraction of the best model during training.**

***Index Terms*—Deep Reinforcement Learning, Deep Deterministic Policy Gradient, DDPG, Twin Delayed DDPG, TD3, Deep Q-Learning, Model Predictive Control**

I. Introduction

As more tasks in industrial environments are automated, industrial logistics and the movement of materials are particular areas of interest and pivotal components in expansive facilities. The operational efficacy of these systems significantly influences production speed, costs, and overall industrial productivity. The solution to these dynamics involves utilizing Autonomous Transport Robots (ATRs) [1] and the intricate dynamics of these robots require the integration of individual controllers and advanced methodologies such as computer vision, sensor fusion, and scheduling.

This study aims to address the challenge of efficient and reliable autonomous robots and the focus centers on real-time obstacle avoidance for mobile robots. Model Predictive Control (MPC) is a tool that can help ensure collision-free navigation of mobile robots [2], which is applicable within industrial setups. While powerful, the nature of MPC implies heavy computation reliance, which can slow down the processes and cause issues with achieving the targets for relevant applications. An important concept is then to provide a good reference for the MPC to take into account when calculating a robot's path. A relevant goal when providing a reference is to weigh between low computation time and relevance such that the reference still provides a good starting point for the MPC.

Motivated by the issue of efficient reference trajectory generators, this study aims to build upon observations made in previous research [3] [4] where Deep Reinforcement Learning (DRL) exhibits significant advancements in addressing obstacle-avoidance situations, prompting to explore the integration of DRL and MPC for ATRs. Through simulations of obstacles and evaluation in accordance with the previous results, this study aims to implement and evaluate the algorithms Deep Deterministic Policy Gradient (DDPG) and Twin Delayed DDPG (TD3).

II. Previous work

This project aims to further develop the ideas of an earlier project [3] within the course SSY226. The previous project investigated the use of DRL, and more specifically Deep Q-Learning (DQL), when planning a trajectory for an ATR while avoiding collisions. Since DQL uses a discrete action space [5], 9 different actions were defined. They correspond to a linear combination of either no acceleration, minimum acceleration, or maximum acceleration for both acceleration and angular acceleration, according to equation (1).

$$\mathbf{u} \in \{a_{\min}, 0, a_{\max}\} \times \{\alpha_{\min}, 0, \alpha_{\max}\} \tag{1}$$

The resulting Deep Q-Network (DQN), was then compared to an MPC controller with the same task. It was concluded that the MPC controller followed the given ideal path quite well when there were no dynamic or non-convex obstacles involved, although with high computational complexity. The DRL agent followed the ideal path less accurately with lower computational complexity.

Another study [4] was then made, where the previous resulting DQN was combined with an MPC controller. This was done by using only the MPC controller when no obstacles are inbound, and switching to using the output of the DQN as a reference to the MPC controller when objects are blocking the original path. The concept is shown in Fig. 1. The combination of DRL and MPC resulted in an agent that had a higher success rate when avoiding obstacles than previous agents, while achieving a lower computational complexity than pure MPC.

[1]Department of Electrical Engineering, Chalmers University of Technology, 41296 Gothenburg, Sweden {**adambu, ilyaku, krimat, nymanga, arvpete, wiselll**}**@chalmers.se**

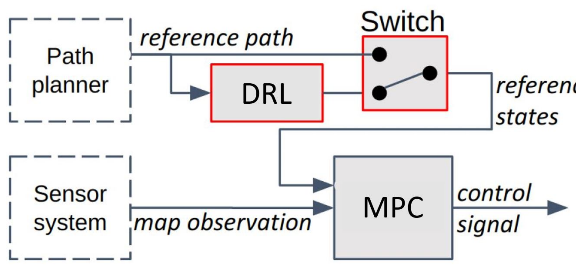

Fig. 1: Concept of DRL/MPC hybrid system.

III. Problem description

The project aims to extend previous studies by implementing DDPG and TD3 as an option to DQN. In order to implement the algorithms, the DRL environment must be adapted to allow a continuous action space. Appropriate reward functions and hyperparameters for the algorithms will be chosen. Furthermore, additional changes to the previous implementations will be performed if deemed to enhance performance. One example is to implement Prioritized Experience Replay (PER) to decrease the effect of catastrophic forgetting and create more stable training. Finally, the resulting DRL agents will be combined with MPC as described by [4] and the corresponding simulations for evaluation will be run. Performance metrics from the simulations will then be used to evaluate the changes made within this project, by comparing to previous results.

IV. Theory

The project includes implementation of the algorithms DDPG and TD3. The models will be trained to output a reference trajectory which is used as an input for the MPC controller.

A. DDPG

One of the DRL algorithms that was investigated in the project is DDPG [6]. It was created by combining the ideas of DQL and an actor-critic approach [7] to reach an algorithm that uses a continuous action domain. The definition can be found below in Algorithm 1. Unlike DQN, DDPG works in continuous action space, which allows for unlimited action options without needing to discretize the action space to 9 options as in [3]. This will allow the controller to be more flexible.

B. TD3

The other DRL algorithm, which will be investigated and compared to DDPG, is TD3. TD3 is an algorithm built upon DDPG, which decreases the risk of overestimation [8]. Value-based DRL methods such as DQL are known to overestimate Q-values and therefore create sub-optimal policies. The same problem with overestimation bias can be observed when DDPG is used. TD3 solves this by utilizing two Q-functions and switching between the two to use the smaller Q-value. The definition can be seen in Algorithm 2.

C. MPC

MPC is an iterative constrained-based control approach that considers a predefined number of time steps in the future, referred to as the horizon N, and calculates the optimal control policy based on minimizing the penalties of the future states and control actions. This is combined with constraints on the states and control based on predefined sets for each quantity such that the control policy doesn't violate these constraints. The optimal control policy for future time steps $k = 0 : N-1$ is calculated by minimizing the objective function

$$\min_{\mathbf{u}_{0:N-1}} J_N + \sum_{k=0}^{N-1} ||\mathbf{x}_k - \tilde{\mathbf{x}}||^2_{\mathbf{Q_x}} + ||\mathbf{u}_k - \tilde{\mathbf{u}}||^2_{\mathbf{Q_u}} \tag{2}$$

where J_N denotes the terminal state cost, $\mathbf{x}_k$ and $\mathbf{u_k}$ the state of the system at time step k and the control signal at time step k, $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{u}}$ the references for the state and control signal, $\mathbf{Q_x}$ and $\mathbf{Q_u}$ penalty matrices for the state and control signal, respectively.

Algorithm 1 DDPG algorithm [6].

```
Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor
  $\mu(s|\theta^\mu)$ with weights $\theta^Q$ and $\theta^\mu$.
Initialize target networks $Q'$ and $\mu'$ with weights
  $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$.
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 $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to
          the current policy and exploration noise.
        Execute action $a_t$ and observe reward $r_t$ and
          observe new state $s_{t+1}$.
        Store transition $(s_t, a_t, r_t, s_{t+1})$ in $R$.
        Sample a random minibatch of $N$ transitions
          $(s_i, a_i, r_i, s_{i+1})$ 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:
            $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i,a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_{s_i}$
        Update the target networks:
            $\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$
            $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$
```

Algorithm 2 TD3 algorithm [8].

```
Randomly initialize critic networks
  $Q_1(s,a|\theta_1^Q), Q_2(s,a|\theta_2^Q)$ and actor-network $\mu(s|\theta^\mu)$
  with weights $\theta_1^Q, \theta_2^Q$ and $\theta^\mu$.
Initialize target networks $Q_i'$ and $\mu'$ with weights
  $\theta_i^{Q'} \leftarrow \theta_i^Q$ and $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer $R$.
for episode = 1, T do
    Select action with exploration noise $a \sim \mu(s) + \epsilon$,
    $\epsilon \sim N(0,\sigma)$ and observe reward r and new states s'
    Store transition tuple $(s,a,r,s')$ in $R$
    Sample mini-batch of N transitions $(s,a,r,s')$ from
    $R$
    $\bar{a} \leftarrow \mu'(s') + \epsilon$, $\epsilon \sim \text{clip}(\mathcal{N}(0,\bar{\sigma}), -c, c)$
    $y \leftarrow r + \gamma min_{i=1,2} Q_i'(s', \bar{a}|\theta_i^{Q'})$
    Update critic by minimizing the loss:
        $$L = \frac{1}{N}\sum_j (y_j - Q_i(s_j, a_j|\theta_i^Q))^2$$
    if episode $mod$ d then
        Update $\theta^\mu$ by the deterministic policy gradient:
        $$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_i \nabla_a Q_1(s,a|\theta^Q)_{|s=\mu(s)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)|_s$$
        Update the target networks:
        $$\theta_i^{Q'} \leftarrow \tau\theta_i^Q + (1-\tau)\theta^{Q'}$$
        $$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
```

V. METHOD

A. Environment

To allow training of the models using DRL, a virtual environment was set up to emulate the dynamics of a real ATR and a real-world environment.

To model the ATR dynamics the discrete state space model seen in equation (3) was used.

$$\underbrace{\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ v_{k+1} \\ \theta_{k+1} \\ \omega_{k+1} \end{bmatrix}}_{\mathbf{x}_{k+1}} = \underbrace{\begin{bmatrix} x_k + T_s v_k \cos\theta_k \\ y_k + T_s v_k \sin\theta_k \\ v_k + T_s a_k \\ \theta_k + T_s \omega_k \\ \omega_k + T_s \alpha_k \end{bmatrix}}_{f(\mathbf{x}_k, \mathbf{u}_k)} \tag{3}$$

The state vector **x** contains both the ATR's position (x, y) and speed v as well as the ATR's direction angle θ and angular speed ω. The ATR's position is obtained by integrating the velocity using the time step T_s while accounting for the direction angle of the ATR, whereas the speed is obtained by integrating the acceleration a, which is one of the two inputs to the system. Similarly, the direction angle is obtained by integrating the angular velocity, which in turn is obtained by integrating the angular acceleration α, which is the second input. The speed and angular velocity are bounded in ranges $[-0.5, 1.5]$ and $[-0.5, 0.5]$ respectively.

The ranges in which the inputs can lie are $[-1, 1]$ for the linear acceleration and $[-3, 3]$ for the angular acceleration. As the DRL algorithms used in this project utilize continuous action space, the input can be represented as

$$\mathbf{u} = [-1, 1] \times [-3, 3] \tag{4}$$

The continuous action space required for the two DRL algorithms is defined utilizing the class *spaces.Box* from the OpenAI Gym framework.

B. Observations

The observation components deemed suitable for the DRL agent to interpret from the environment are:

- *Speed observation* - observes ATR's speed.
- *Angular velocity observation* - observes ATR's angular velocity.
- *Reference path corner observation* - observes the distance and relative angle to upcoming corners along the reference path.
- *Reference path sample observation* - observes the distance and relative angle to equally spaced upcoming points along the reference path.

Additionally, in order for the agent to detect approaching obstacles, two different types of surrounding information acquisition were tested:

1) *Image observation* - observes images of the environment surrounding the ATR,
2) *Ray and sector observation:*
   a) *Ray observation* - uses a LiDAR to cast rays from the ATR against all obstacles in the environment and observes the length of these rays.
   b) *Sector observation* - divides the space around the ATR and reports the distance to the closest obstacle within each sector.

C. Rewards

Finding a good reward function was a significant part of tuning the DRL models to acquire the best performance. Generally, the reward function is very similar to the one in [3]. However, since the action space is continuous in this project, jerk rewards were also added to the function in order to prevent the ATR from chaotic jerky behavior and improve the smoothness of the inputs. The reward function consists of these components:

$$R_k^{\text{collision}} = \begin{cases} -\lambda^{\text{collision}}, & \text{collision at time-step k} \\ 0, & \text{otherwise} \end{cases} \tag{5a}$$

$$R_k^{\text{goal}} = \begin{cases} \lambda^{\text{goal}}, & \text{goal reached at time-step k} \\ 0, & \text{otherwise} \end{cases} \tag{5b}$$

$$R_k^{\text{speed}} = -\lambda^{\text{speed}} T_s \text{max}(0,\text{sign}(v_{\text{ref}})(v_k - v_{\text{ref}})) \tag{5c}$$

$$R_k^{\text{path}} = \lambda^{\text{path}}(l_k - l_{k-1})^2 \tag{5d}$$

$$R_k^{\text{cte}} = -\lambda^{\text{cte}} T_s (d_k^p)^2 \tag{5e}$$

$$R_k^{\text{jerk}} = -\lambda^{\text{jerk}}(a_k - a_{k-1})^2 \tag{5f}$$

$$R_k^{\text{angular jerk}} = -\lambda^{\text{angular jerk}}(\alpha_k - \alpha_{k-1})^2 \tag{5g}$$

It is important to highlight that in equation (5c) the speed factor is linearly dependent on the path factor, $\lambda^{\text{speed}} = 2\lambda^{\text{path}}$. In equation (5e) it is noted that the cross-track error (CTE) reward depends on $(d_k^p)^2$, representing the squared CTE which is the Euclidean distance between the agent's current position and the closest point on the path at time step k.

The final reward function is

$$R = R^{\text{collision}} + R^{\text{goal}} + R^{\text{speed}} + R^{\text{path}} + R^{\text{cte}} + \\ + R^{\text{jerk}} + R^{\text{angular jerk}} \tag{6}$$

D. Implementation of Algorithms

The code for this project is available in an online repository [1]. Both of the DRL algorithms that were used to train the agent were implemented using *Stable-Baselines3* which is a *Python* library containing multiple different DRL algorithms from OpenAI. In this case, the algorithms DDPG and TD3 were imported from the library. The output from the networks being a vector of normalized inputs $\mathbf{u} = [\bar{a}, \bar{\alpha}]$, with $\bar{a} \in [-1, 1]$ and $\bar{\alpha} \in [-1, 1]$. $\bar{\alpha}$ was then scaled by 3 before being used as the input to the ATR dynamics as the ATR uses the range $\alpha \in [-3, 3]$.

The neural networks of the two algorithms were then trained for 10 000 000 iterations using different maps, with randomly generated obstacles.

When training, the model is evaluated on the map shown in Fig. 2 every N_{eval} time step. If the reward turns out to be higher than the previously best model, that model is stored as the best model. This ensures that the best model has been stored even if the best model is not the last iteration of the model or if the training stops early.

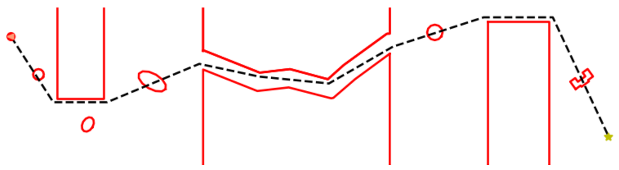
Fig. 2: Map used for finding the best model

[1]https://github.com/adamburman/TrajTrack_MPCnDQN_RLBoost_Design_Project_Group_1

E. Prioritized Experience Replay

In order to address the issue of inefficient sample utilization the technique known as Prioritized Experience Replay (PER) [9] was implemented. The utilization of PER could reduce the training time and improve the stability of the training process. PER allows for more frequent replay of observations associated with both successful and unsuccessful performances. Therefore, both the DDPG and TD3 algorithms from the *Stable-Baselines3* library were extended to leverage PER for more efficient and effective training.

To handle the prioritized sampling a custom replay buffer using a sum-tree data structure was added. One of the previous studies [3] also suggests that PER could help remedy the issue of catastrophic forgetting, which is when the model overwrites useful features during training.

F. Training data

The training scenarios used in this study were heavily inspired by the previous study [3]. Three types of scenarios or maps were used as this proved to be good in previous studies. This ensures that the agent is exposed to multiple versatile environments which will help it to generalize. Fig. 3 shows an example for each type of map.

The first type is map a) in Fig. 3. It consists of three big rectangular objects that are randomly placed (with some constraints). A reference path is then generated using the A* algorithm [10]. Smaller obstacles are then randomly placed on the map. These consist of moving ellipses that move according to a predefined randomly generated straight path. They can also be static non-convex objects in a U- or L-shape. The distribution between convex and non-convex objects is random and is on average 50% of each.

The second type is map b) in Fig. 3. It consists of a corridor with three randomized turns. This is implemented because previous studies found that, without these kinds of scenarios during training, the agent tends to perform badly when presented with narrow openings and corridors.

An example of the third type can be seen as (c) in Fig. 3 which comes from [11]. This is eleven manually created maps that represent some realistic scenarios.

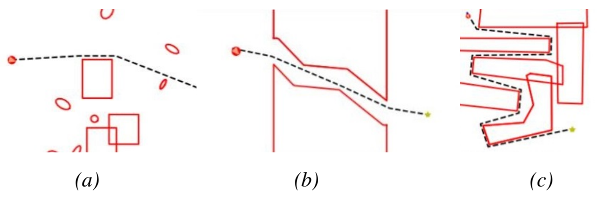
(a) (b) (c)

Fig. 3: Samples of maps used for training

VI. Evaluation

In order to fairly compare the performance of the DDPG and TD3 models to previous results, the same evaluation used in [4] was implemented. The models were then evaluated by navigating a predetermined series of maps with different types of obstacles. Each map was used 50 times and performance metrics were collected each time to achieve average scores. Metrics were collected for the two DRL algorithms, a pure MPC agent, and the hybrid agents which combine DRL and MPC. The DRL and hybrid agents were also evaluated using both LiDAR and visual inputs.

A. Obstacles

The predetermined maps created for evaluation were divided into two scenes. The first scene consists of four types of obstacles, with options for size. The first three types can be seen in Fig. 4, and the fourth can be seen in Fig. 5.

1) One rectangular obstacle in the middle of the planned path, where a) is smaller, and b) is larger.
2) Two rectangular obstacles in a staggered position, creating a corridor in between. In c), the first object is smaller, and in d) it is larger.
3) One U-shaped non-convex object in the middle of the planned path, where e) is shorter, and f) is longer.
4) Two circular dynamical objects moving in two different directions as shown in Fig. 5

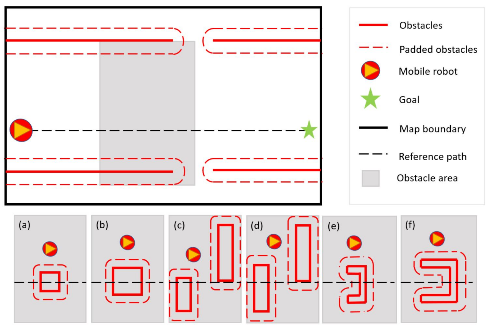

Fig. 4: Six of the test cases for scene 1 [4].

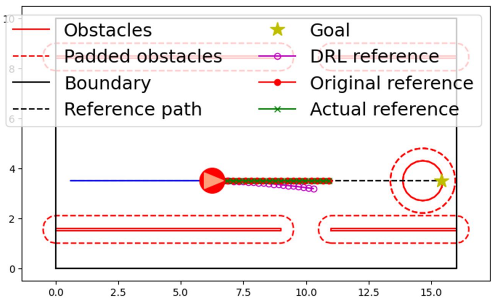

Fig. 5: The last test case for scene 1.

Scene 2 consists of three turns, where an obstacle is placed on the reference path. This type of map was added since previous agents were observed to have problems with this type of map. The three different maps can be found in Fig. 6.

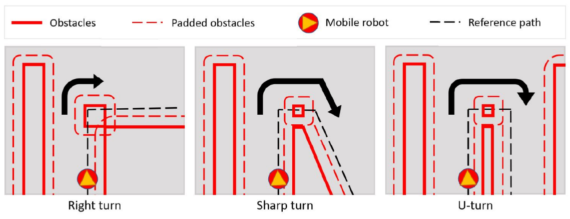

Fig. 6: The tree test cases for scene 2 [4].

B. Performance metrics

A number of metrics were used by [4] to enable a thorough analysis of the different agents' performance. All metrics are averaged over 50 simulations.

- Computation time. Averaged over time steps to enable comparisons between models that don't finish simulations equally fast. Presented by mean, maximum, and median computation times.
- Deviation (distance to the reference path). Without obstacles on the reference path, a small value would be ideal. Since the obstacles are of different sizes, the metric will only be useful when comparing different models evaluated on the same map.
- Action smoothness. The second derivative of the action. A small value corresponds to a smoother trajectory.
- Clearance (closest distance to obstacles). Also depends heavily on the map. If a corridor is on the map, a smaller value would be reasonable.
- Finish time. Time steps needed to reach the goal. A finish time of over 200 time steps is considered a failed attempt.
- Success rate. The percentage of successful simulations out of 50. Results with under 30% success rate are not included.

C. Presentation of results

The resulting performance metrics from the 50 simulations of each scenario for every agent can be found in table I. In the table, the different agents are specified as MPC, {DRL algorithm}-L, {DRL algorithm}-V, H-{DRL algorithm}-L, and H-{DRL algorithm}-V. The one called MPC is pure MPC to enable comparison with the other agents and the previous results from [4]. The last part of the other names -L and -V represent LiDAR and camera (vision) inputs respectively. The prefix H- represents the hybrid agents that combine DRL and MPC, and the ones without the prefix correspond to pure DRL.

TABLE I: Performance metrics collected from the simulations of different agents. Averaged over 50 simulations. Results with under 30% success rate are not shown.

Scene	Obstacle type	Method	Computation time (ms/step)			Deviation (m)		Action smoothness		Other		
			mean	max	median	mean	max	speed	angular speed	clearance (m)	finish time step	success rate (%)
Scene 1	Rectangular obstacle (a-medium)	MPC	63.99	303.35	39.25	0.41	1.55	0.04	0.04	0.72	71	100
		DDPG-L	0.22	0.33	0.21	0.87	1.84	0.03	0.05	0.64	117	100
		DDPG-V	0.67	1.03	0.63	0.95	1.90	0.05	0.04	0.86	136	100
		H-DDPG-L	40.25	356.49	18.69	0.83	2.25	0.02	0.06	0.84	79	100
		H-DDPG-V	36.61	304.47	19.16	1.07	2.54	0.02	0.04	1.04	77	100
		TD3-L	0.24	0.44	0.22	0.76	1.55	0.04	0.05	0.51	126	68
		TD3-V	0.69	1.06	0.64	0.8	1.77	0.05	0.02	0.76	133	100
		H-TD3-L	40.11	387.22	18.85	0.84	2.21	0.02	0.05	0.84	82	100
		H-TD3-V	32.33	126.82	18.11	0.63	1.84	0.02	0.03	0.85	76	100
	Rectangular obstacle (b-large)	MPC	62.26	261.57	38.71	0.53	1.91	0.04	0.05	0.67	76	100
		DDPG-L	0.22	0.33	0.21	1.19	2.73	0.04	0.04	0.53	132	100
		DDPG-V	0.67	0.99	0.63	1.03	2.11	0.05	0.04	0.79	135	100
		H-DDPG-L	40.02	346.1	18.50	1.05	2.94	0.02	0.05	0.8	85	100
		H-DDPG-V	35.74	263.31	18.99	1.12	2.69	0.02	0.03	1.02	78	100
		TD3-V	0.67	1.05	0.63	0.89	1.83	0.05	0.03	0.71	130	100
		H-TD3-V	34.13	132.57	20.21	0.69	1.97	0.02	0.03	0.79	76	100
	Two obstacles (c-small stagger)	MPC	74.99	377.12	63.0	0.43	1.28	0.05	0.04	0.68	71	100
		DDPG-V	0.70	1.49	0.64	0.78	1.83	0.06	0.05	0.50	162	42
		H-DDPG-L	51.84	240.39	42.84	0.84	2.19	0.04	0.05	0.68	99	100
		H-DDPG-V	57.54	319.85	36.9	0.78	1.82	0.03	0.04	0.72	92	100
		H-TD3-L	54.38	186.63	38.8	0.86	2.08	0.03	0.03	0.71	96	100
		H-TD3-V	68.3	957.61	40.8	0.7	1.8	0.03	0.04	0.7	91	100
	Two obstacles (d-large stagger)	DDPG-V	0.69	1.54	0.63	1.03	2.37	0.06	0.04	0.51	177	44
		H-DDPG-L	56.38	321.31	50.26	1.14	3.33	0.03	0.04	0.70	110	100
		H-DDPG-V	53.31	274.10	41.94	0.94	2.39	0.03	0.04	0.70	91	100
		H-TD3-V	97.77	965.95	51.99	1.29	3.39	0.05	0.06	0.63	103	74
	U-shape obstacle (e-small)	DDPG-L	0.23	0.43	0.21	1.07	2.3	0.03	0.05	0.65	117	100
		DDPG-V	0.22	0.35	0.21	1.09	2.32	0.03	0.05	0.64	117	100
		H-DDPG-V	33.04	157.78	19.50	1.15	2.71	0.02	0.04	1.24	77	100
		TD3-L	0.23	0.4	0.22	0.61	1.33	0.03	0.04	1.9	121	76
		TD3-V	0.67	1.01	0.63	0.5	1.39	0.05	0.04	1.73	132	64
		H-TD3-L	26.83	212.97	16.63	0.49	1.77	0.01	0.03	1.81	79	100
	U-shape obstacle (f-large)	DDPG-V	0.67	1.00	0.62	1.17	2.53	0.05	0.05	0.65	141	100
		H-DDPG-L	39.75	581.66	23.28	1.21	3.01	0.02	0.03	0.81	79	100
		H-DDPG-V	37.99	450.42	26.45	1.33	3.06	0.02	0.04	0.81	79	100
		TD3-V	0.67	1.02	0.63	0.84	1.92	0.05	0.03	0.73	134	100
		H-TD3-V	37.73	329.92	20.04	1.01	2.54	0.02	0.04	0.85	84	100
	Dynamic obstacle (face-to-face)	MPC	413.68	>1000	18.87	0.30	1.20	0.06	0.11	1.20	121	60
		DDPG-L	0.22	0.29	0.21	0.74	1.47	0.02	0.06	1.90	115	57
		DDPG-V	0.67	1.03	0.63	0.67	1.33	0.04	0.04	1.72	168	46
		H-DDPG-L	29.18	219.60	16.79	0.49	1.76	0.01	0.03	1.90	76	100
		H-DDPG-V	22.94	178.51	16.05	0.65	2.22	0.02	0.02	1.88	78	100
		TD3-L	0.23	0.4	0.22	0.61	1.33	0.03	0.04	1.9	121	76
		TD3-V	0.67	1.01	0.63	0.5	1.39	0.05	0.04	1.73	132	64
		H-TD3-L	26.83	212.97	16.63	0.49	1.77	0.01	0.03	1.81	79	100
Scene 2	Right turn with an obstacle	**H-DDPG-L**	51.84	351.49	30.74	0.48	1.79	0.02	0.03	0.73	134	86
		H-DDPG-V	36.55	267.18	16.39	0.53	1.84	0.02	0.03	0.81	134	100
		H-TD3-L	42.68	253.04	21.46	0.48	1.75	0.02	0.03	0.72	137	86
		H-TD3-V	27.69	110.23	15.19	0.46	1.92	0.01	0.01	0.78	130	100
	Sharp turn with an obstacle	**H-DDPG-L**	26.22	138.11	15.88	0.38	1.56	0.01	0.01	0.82	146	100
		H-DDPG-V	29.01	227.07	16.06	0.37	1.14	0.01	0.02	0.84	149	100
		H-TD3-L	24.49	129.61	15.09	0.35	1.28	0.01	0.01	0.82	146	100
		H-TD3-V	20.0	261.67	14.69	0.36	1.39	0.01	0.01	0.85	148	100
	U-turn with an obstacle	**H-DDPG-L**	35.42	446.89	15.91	0.41	1.45	0.01	0.01	0.74	145	100
		H-DDPG-V	25.19	195.36	15.56	0.35	1.23	0.01	0.01	0.79	146	100
		H-TD3-L	28.99	305.86	15.25	0.37	1.23	0.01	0.01	0.78	145	100
		H-TD3-V	23.41	298.37	14.68	0.35	1.33	0.01	0.01	0.78	145	100

VII. Discussion

A. Reward function and reward factors

Given the limited amount of training sessions, the possibility of evaluating suitable reward factors was restricted. Consequently, we were unable to conclude if there are any other terms that need to be added to the reward function.

B. Scenarios and DRL agents

After initial testing, it became clear that the best results were reached when models using PER were evaluated, compared to the same models without PER. This was the case for both DDPG and TD3 and therefore the results in table I are from models using PER.

It can be seen in table I that the DDPG model outperformed the TD3 model in almost every scenario, which was not the expected result as TD3 is based on DDPG. It would therefore be expected that the two models perform similarly. The better performance of DDPG compared to TD3 indicates that overestimation bias, that TD3 should decrease, is not a prominent issue compared to other problems during training. More trained models could also be trained and evaluated to better asses the algorithm's overall performance, since there is a nondeterministic component that influences the models during training.

A common problem with reinforcement learning is catastrophic forgetting where the training overwrites previously learned beneficial behaviour. Which can result in the reward going down between iterations from a previously higher value. However, as the evaluation during training in this case was done using the collected rewards on a single map, the training results do not show a complete picture of the performance. An example of training results can be seen in Fig. 7. Therefore it is difficult to determine whether the decreased score during training depends on catastrophic forgetting or that other useful features are learned that happen to give worse results on the specific evaluation map.

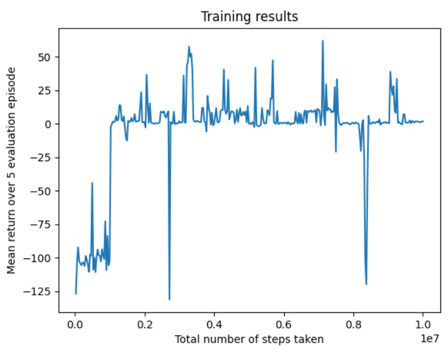

Fig. 7: Mean reward during training for DDPG with LiDAR inputs.

C. Extracting the best model

The currently implemented method of saving the best model is to evaluate the model on a specific map periodically during training and save the model with the highest reward on that map. By looking at Fig. 7 it can be seen that this would correspond to the highest peak.

This method could be improved since the evaluation only considers a specific set of features of the agent that is useful in this map. The resulting model will then tend to be biased to the evaluation map regardless of how diverse the training data is. To slightly lower the impact of this, the map was redesigned to incorporate multiple different kinds of obstacles. However, this does not solve the problem.

One way to get rid of this bias is to use randomized maps for the evaluation as well. However, the problem is that a randomized evaluation map would instead make the comparison to previous models during training infeasible (if the comparison metric is the total reward). Another option is to instead use a large number of maps. However, that would instead increase the training time substantially which is not desirable either.

This is a problem that needs to be addressed since testing revealed that the model that is saved during training is not always the model that performs the best during testing. This was noticed by saving the model at multiple time steps during training and comparing it to the one that the code deems to be the best. Currently, the method used only guarantees that the model performing the best on that specific map is saved, not necessarily the one that is the best overall.

D. Comparison to previous work

When comparing the results in table I with the results using the DQN algorithm in [4], it is clear that the best model within this project outperforms previous results. It can be seen that the hybrid DDPG model with camera input performs best in terms of success rate since it has a 100 % success rate in all scenarios. The corresponding model with LiDAR inputs received 100 % success rate in all scenarios except the *Right turn with an obstacle* where it had an 86 % success rate and *U-shape obstacle (e-small)* got below 30 %. This means that the DDPG hybrid with the camera input performs as well or better than the DQN hybrid using the camera as input when looking at the success rate. The same goes for the LiDAR input for all scenarios except *Right turn with an obstacle*.

It can also be seen that the action smoothness for the pure DDPG and TD3 models is about 10 times smoother than the pure DQN models used in [4]. This is a promising result as one of the main purposes of switching to models utilizing a continuous action space instead of a discrete action space is to allow for smoother actions from the ATR. The increase in smoothness was achieved by a combination of using a continuous action space, and penalizing jerk in the movement of the ATR.

The computation time of the different agents can't be directly compared to the results [4] since it has not been evaluated on the same hardware. However, since the pure MPC implementation is the same, the computation time relative to the MPC can be used to draw conclusions. The computation times for the pure DRL models are still low compared to the MPC, making them an insignificant part of the computation time for the hybrid agents. The mean and median computation times of all hybrid models are lower than the corresponding time for MPC, in all cases when both agent types finish the evaluation. In the previous study, there as some cases where this is not true.

VIII. Conclusion

In conclusion, it proved possible to implement DRL algorithms that use a continuous action space for this application. It can achieve the same or better success rate as the discrete action space DRL algorithm while also having a higher action smoothness. The computation time of the hybrid models is also kept lower than when using pure MPC. Using PER for the training of the algorithms can give better results. DDPG outperformed TD3 for this task but no definite conclusion can be made that it is a better choice. This is because that would require more thorough tuning and training and a more in-depth analysis comparing the results of the two algorithms.

IX. Future work

As stated above, the evaluation map that was used during training was not the most effective at extracting the best model. Therefore, this can be revised to achieve better results. One of the ways to do it is to use several evaluation maps with numerous turns, tunnels, static and dynamic obstacles in different configurations, to evaluate more features of the agent. Measuring score in other ways than reward might also be useful.

It would also be useful to revise and optimize the training process, including tuning the hyperparameters, eg. architecture size, in order to improve performance and reduce the training time, as it was found to be high for the used DRL models.

Another thing to consider in the future would be to revise the reward function and explore if any other terms should be added to it. To be more specific, it would be useful to conduct a more accurate tuning of the jerk rewards such that linear and angular acceleration get smoothened without penalizing the movement of the ATR.

The last thing that would be good is to explore how the resolution of the image observation affects the performance, as it has been noted that when narrow corridors are observed under an oblique angle, they tend to seem narrower because of the pixelization.

X. Acknowledgements

We would like to extend our heartfelt gratitude to our supervisors, Kristian Ceder and Ze Chang. Their support and advice throughout the project were immensely valuable. It kept the project on track by giving us a good understanding of the background and helping to solve issues quickly while keeping us motivated and interested.

References

[1] H. Andreasson, A. Bouguerra, M. Cirillo, D. N. Dimitrov, D. Driankov, and L. Karlsson, "Autonomous transport vehicles: Where we are and what is missing," *IEEE Robotics & Automation Magazine*, vol. 22, no. 1, pp. 64–75, 2015. DOI: 10.1109/MRA.2014.2381357.

[2] A. Salimi Lafmejani and S. Berman, "Nonlinear mpc for collision-free and deadlock-free navigation of multiple nonholonomic mobile robots," *Robotics and Autonomous Systems*, vol. 141, 2021. DOI: https://doi.org/10.1016/j.robot.2021.103774.

[3] K. Ceder, A. Enliden, O. Eriksson, S. Kylander, and R. Sridhara, "Collision free trajectory planning using deep reinforcement learning," *SSY262 2022*, 2022.

[4] Z. Zhang et al., "Collision-free trajectory planning of mobile robots by integrating deep reinforcement learning and model predictive control," in *CASE*, IEEE, 2023.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, and Y. Tassa, *Continuous control with deep reinforcement learning*, 2019. arXiv: 1509.02971 [cs.LG].

[7] M. Wu, Y. Gao, A. Jung, Q. Zhang, and S. Du, "The actor-dueling-critic method for reinforcement learning," *Sensors*, vol. 19, no. 7, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/7/1547.

[8] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, J. Dy and A. Krause, Eds., Oct. 2018. [Online]. Available: https://proceedings.mlr.press/v80/fujimoto18a.html.

[9] L. L. e. a. Y. Hou, "A novel ddpg method with prioritized experience replay," IEEE, 2017. DOI: 10.1109/SMC.2017.8122622.

[10] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. DOI: 10.1109/TSSC.1968.300136.

[11] J. Berlin, G. Hess, A. Karlsson, W. Ljungbergh, Z. Zhang, and K. Åkesson, *Trajectory generation for mobile robots in a dynamic environment using nonlinear model predictive control*, Mar. 2021. DOI: 10.36227/techrxiv.14215862.