Robust mobile robot control through adversarial training

Elsa Andersson
Systems, Control and Mechatronics
Chalmers University of Technology
Email: elsaand@chalmers.se

Alice Dalevi
Systems, Control and Mechatronics
Chalmers University of Technology
Email: alicedal@chalmers.se

Thim Högberg
Systems, Control and Mechatronics
Chalmers University of Technology
Email: hthim@chalmers.se

Oskar Höglund
Systems, Control and Mechatronics
Chalmers University of Technology
Email: oskarhog@chalmers.se

Dharunkumar Senthilkumar
Systems, Control and Mechatronics
Chalmers University of Technology
Email: dhasen@chalmers.se

***Abstract*—Robust navigation in dynamic environments is a fundamental requirement for autonomous mobile robots. Traditional reinforcement learning approaches often rely on static or manually crafted environments, which may not sufficiently prepare agents for unpredictable disturbances. This work investigates an adversarial training curriculum where a primary navigation agent is trained against an active adversary designed to interfere with its progress. We evaluate three adversary implementations: a physics-based Linear-Quadratic Regulator (LQR), a Model Predictive Controller (MPC) and a self-play Reinforcement Learning scheme. Our results demonstrate that training against a competent adversary significantly enhances robustness, more than doubling the success rate in dynamic test scenarios compared to a baseline trained on static obstacles (78.8% vs. 32.0%). Furthermore, we find that the complexity of the training environment is critical, specifically, learning-based adversaries require the presence of static constraints to encourage robust navigation strategies. However, despite these improvements in dynamic avoidance, the agents struggle to generalize to precise settings, such as narrow corridors, highlighting the limitations of this approach.**

I. Introduction

To deploy mobile robots in real-world, dynamic environments, robust and safe navigation is essential. A major challenge in developing such robots with Reinforcement Learning (RL) lies in designing training methods that produce resilient policies capable of handling unanticipated situations and dynamic agents. such as other robots or humans.

Conventional simulation-based training often relies on manually crafted environments with static obstacles [1]. Nguyen et al. [1] investigated the use of large language models to predict scenarios that cause undesirable behavior, noting that generating these environments is often time-consuming and lacks flexibility. Furthermore, these methods frequently fail to reflect the complexity and unpredictability of the real world. As a result, generating a sufficiently diverse and challenging set of training scenarios, especially those involving dynamic obstacles, remains a significant challenge.

The purpose of this project is to develop an approach for automatically generating a challenging and adaptive training curriculum. Rather than relying solely on hand-crafted environments, we introduce an adversarial agent whose role is to actively interfere with the primary agent's navigation task. The core hypothesis is that by training the primary agent against "worst-case" dynamic disturbances, it will learn a more robust and generalizable policy for reaching its goals, effectively acquiring the ability to evade dynamic obstacles as a natural consequence of the training process.

We investigate three distinct methods for controlling the adversarial agent

- A pure-pursuit Linear-Quadratic-Regulator (LQR) controller inspired by Xiang et al. [2].
- A Model Predictive Controller (MPC).
- An RL-based controller inspired by Baker et al. [3].

In all configurations, the primary agent is an RL-based navigation policy rewarded for reaching a goal along a predefined path and penalized for collisions. The adversarial agents are designed to challenge this progress: the LQR adversary increases linear acceleration once the primary agent's success rate exceeds a specified threshold, the MPC adversary is optimized for collision, and the RL adversary receives rewards directly opposing those of the primary agent.

Each adversarial method is applied during training, after which the primary agent is evaluated on both a simple environment and a more challenging one. These results are compared to a baseline where the primary agent is trained only on the environments themselves, without any adversarial disturbances.

The main objective of this work is to investigate if incorporating an adversarial agent improves the robustness and success rate of the primary navigation policy compared to standard training.

II. Related Works

One relevant work for this project is the Hide-and-Seek framework by Baker et al. (2020) [3]. In this work, multiple

agents are trained in a shared environment using reinforcement learning and self-play. The agents have opposing goals and continuously adapt to each other during training. This interaction causes the task difficulty to increase automatically over time, leading to more complex behaviors without manually designing a training curriculum.

The main contribution of Baker et al. is showing that adversarial interaction between agents can act as a natural curriculum. As one agent improves, it creates new challenges for the opposing agent, which in turn must adapt. As the agents compete, learning happens in several stages. New strategies appear naturally because the agents must adapt to an improving opponent.

Although the Hide-and-Seek environment focuses on a game-like setting with objects such as boxes and ramps, the underlying idea is not limited to that domain. The key concept is that adversarial agents can be used to automatically generate challenging training scenarios, which motivates the approach taken in this project.

In this project, we use the main idea from the Hide-and-Seek framework in a different context, namely mobile robot navigation. The focus is on training an RL-based navigation controller to handle disturbances from another agent.

We deliberately limit the related work to this paper, as it is directly relevant to the use of adversarial training for RL controllers. Many other design choices and ideas in this project were not taken from existing literature but were instead developed together with the project supervisor based on iterative discussions and experimentation.

III. Background

A. Reinforcement Learning

Reinforcement Learning (RL) is concerned with approximating a policy for which a maximum reward will be received. At each discrete time step t, with the current state $s \in \mathcal{S}$, the agent selects an action $a \in \mathcal{A}$ in accordance with its policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$, and is given a reward r and a next state s' from the environment by $p : (\mathcal{S} \times \mathcal{A}) \rightarrow \Delta(\mathcal{S})$. The return from time t until T is defined as the discounted sum of rewards $R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k(s_k, a_k)$, where the discount factor γ determines the priority of short-term rewards.

The object of RL is then to find an optimal policy π_θ^*, with parameters θ, which maximizes the expected reward $J(\theta) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}[R_{t_0}]$. For continuous control problems, the policy π_θ can be updated through the gradient of the expected return $\nabla_\theta J(\theta)$, and when the model is deterministic we can formulate an actor, critic setup for the network update rules. The policy, also called actor, can be updated through one such method called the Deep Deterministic Policy Gradient (DDPG) algorithm [4][5],

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_i \sim p_\pi}[\nabla_a Q^\pi(s,a)|_{a=\pi(s)} \nabla_\theta \pi_\theta(s)],$$

where $Q^\pi(s,a) = \mathbb{E}_{s_i \sim p_\pi, a_i \sim \pi}[R_t|(s,a)]$ is the expected return from s while following the current policy π, and is called the value function or the critic.

If the state space is comparatively small then Q-learning can be used to learn the value function through temporal difference [6][7], which is based on the bellman equation [8]. The bellman equation describes the relationship between an action a from a state s with its expected return. This can be formulated as,

$$Q^\pi(s,a) = r + \gamma \mathbb{E}_{s',a'}[Q^\pi(s',a')], \quad a' \sim \pi_\theta(s'),$$

and if $\pi_\theta = \pi_\theta^*$ this will be the optimal return policy.

For large state spaces (i.e. continuous states), the Q-function can not be realized exactly but can be approximated using a differentiable function approximator $Q_\phi(s,a)$ with network parameters ϕ. In deep Q-learning [9], the network is updated via temporal-difference learning using a secondary, frozen target network $Q_{\phi'}(s,a)$ to provide a stable objective over multiple updates,

$$Q_\phi(s,a) = r + \gamma Q_{\phi'}(s',a'), \quad a' \sim \pi_{\theta'}(s'),$$

where actions are selected according to a target actor network $\pi_{\theta'}$. The target network parameters are updated either periodically to exactly match the current network, or incrementally using a soft update by $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$. These updates can be performed off-policy by sampling random mini-batches of transitions from an experience replay buffer [10].

B. TD3

Twin Delay Deep Deterministic Policy Gradient (TD3) [11] is a reinforcement learning algorithm designed for continuous control tasks. It builds upon the DDPG method but introduces improvements to make training more stable. Specifically it addresses DDPG's tendency to overestimate value functions. To solve this TD3 uses three strategies.

Firstly, it uses two separate critic networks to evaluate actions and the minimum value is used to prevent overestimation. Secondly, the policy network is updated less frequently than the value networks, allowing value estimates to stabilize first. Finally, it adds noise to target actions during training to smooth the value function.

C. Pure pursuit

The pure pursuit algorithm is a geometric path-tracking method that computes the curvature required to drive a vehicle from its current pose to a goal point located at a look-ahead distance ℓ along a reference path, shown by Couter et al. [12]. Using a unicycle kinematic model, let the agent pose in the global frame be $p_v = (x_v, y_v, \varphi)$ and let the chosen look-ahead (goal) point on the path be p_g. Transform the goal into the agent (body) frame via

$$\begin{bmatrix} x_g \\ y_g \end{bmatrix} = R(-\varphi)(p_g - (x_v, y_v)), \quad R(\varphi) = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix},$$

and using the transformed goal position Couter showed that the following relationship for the curvature of the arc can be formed. Let r be the radius of the arc which is tangent along the agents forward direction and passes through the goal, then

$$r = \frac{\ell^2}{2x_g},$$

and for a given forward velocity v_f yields a reference angular velocity ω_{ref} which should be tracked. For our purposes we discarded the notion of a path trajectory and simply considered the goal point to always be the position of the primary agent.

D. LQR

The Linear Quadratic Regulator (LQR) is an optimal control strategy for linear dynamical systems described by the state-space equation $\dot{x} = Ax + Bu$. The objective of LQR is to find a control law $u(t)$ that minimizes the quadratic cost function:

$$J = \int_0^\infty (x^T Q x + u^T R u) dt$$

where Q and R are weight matrices that penalize state deviations and control effort respectively. The optimal control law is given by state feedback $u = -Kx$, where the gain matrix K is derived from the solution P to the Algebraic Riccati Equation:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$

with $K = R^{-1}B^T P$.

E. MPC

Model Predictive Control (MPC) computes control actions by predicting the system's future behavior over a finite horizon using a dynamics model. At each timestep, candidate control sequences are evaluated by simulating their resulting trajectories and scoring them with a task-specific cost function. The first control input of the best-performing sequence is applied, and the optimization is repeated in a receding-horizon fashion.

For robotic systems with nonlinear dynamics and real-time constraints, sampling-based MPC variants are commonly used. These approaches avoid solving a full continuous optimization problem by evaluating a finite set of sampled control sequences, retaining the predictive planning capability of MPC while remaining computationally tractable.

IV. Method

A. Environment structures

The primary agent was trained across a series of environments designed with varying levels of complexity. These configurations, which serve as the basis for the comparative evaluation, are detailed below.

1) Static Obstacles Only: This setting contains 1 to 4 randomly placed static rectangular obstacles.

2) Train 1: To expose the primary agent to dynamic environments without introducing the adversarial agent, the Train 1 map, shown in Fig. 2, includes both static and dynamic obstacles. Three static square obstacles: a medium block, a large block and a U-shaped non-convex obstacle. These create structural constraints in the environment. In addition, two dynamic obstacles are present: a vertical sweeper and a horizontal sweeper. The obstacles are placed randomly. These obstacles introduce time-varying hazards and require the agent to adapt its trajectory in real time.

3) Adversarial: This environment introduces a dynamic adversarial agent in an open workspace without static obstacles. The adversary actively interferes with the primary agent's navigation. To prevent overfitting to a specific geometry, the adversary's size is randomized at the start of each episode.

4) Adversarial Obstacles: Building upon the Static Obstacles environment, this configuration (Fig. 1) introduces one to four randomly positioned rectangular obstacles alongside an adversarial agent. To prevent the policy from overfitting to a specific geometry, the adversary's dimensions are randomized across episodes.

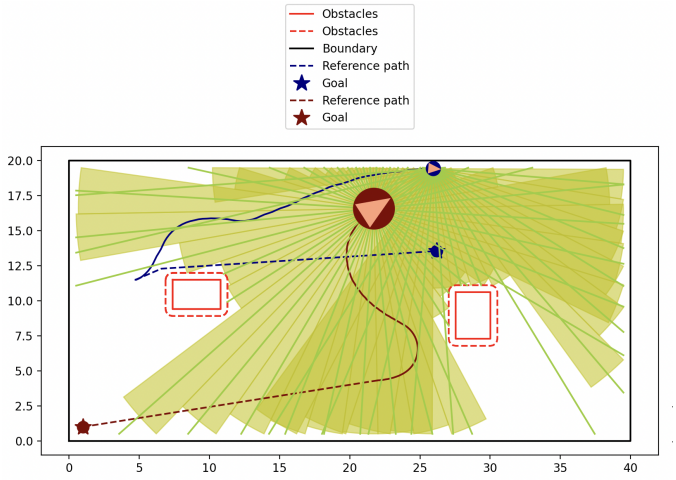


Fig. 1. The Adversarial Obstacles environment featuring random static obstacles and an adversary. The Adversarial environment is identical but lacks static obstacles, while the Empty environment contains neither. The Static Obstacles Only environment mirrors this setup but excludes the adversary. In this figure, the primary agent is represented by a blue circle with an internal triangle, whereas the adversarial agent is represented by a red circle with an internal triangle. Finally, the dashed lines represent the reference path, while the solid lines represent the actual path.

5) Observations: An agents observations are divided into internal and external, where internal observations are concerned with the agent pose, the goal position and the path, and the external observations are concerned with the ray and segment lengths. One unique observation vector is generated for each agent by the environment, and the primary agent can always only access its own vector. The external observations are stored for 2 consecutive time steps to allow agents to infer obstacles velocities.

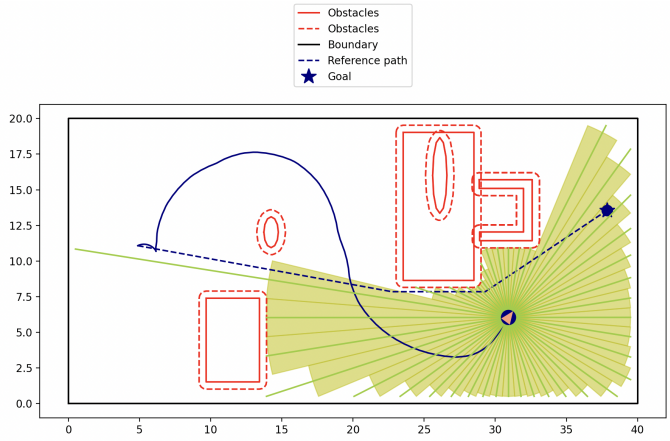


Fig. 2. The Train 1 map featuring static and dynamic obstacles. The dashed lines represent the reference path, while the solid lines represent the actual path.

B. Base case

The Base Case serves as the benchmark against which all subsequent control solutions are evaluated. No adversarial agent is present. Instead, the primary agent learns solely from a reward function composed of three components:

- A collision penalty to discourage contact with walls or obstacles.
- path-progress reward scaled by a factor, which provides dense shaping feedback based on the agent's progress along the reference path and penalizes divergence.
- A goal-reaching reward, encouraging successful task completion.

The base case is trained on two environments, *Static Obstacles Only* and *Train 1 Map*, ensuring exposure to purely static as well as mixed static–dynamic scenarios. This produces a robust baseline policy suitable for comparison with advanced controllers and adversarial training schemes.

C. Primary policy implementation

The primary agent's policy specification is the same for all training methods, and uses the TD3 algorithm for policy optimization. The internal structure of the network is an input layer which receives the observation vector, 2 MLP layers with 256 nodes each which uses the ReLU activation function, and an output layer with 2 nodes which is passed through a tanh sigmoid to ensure the output limits $[-1, 1]$ are preserved. The input observation vector consist of the internal agent pose, and the external rays and segments. The internal agent pose at time t_k is stored only for timestep k, and then discarded for the new pose at time t_{k+1}. The external rays and segments are stored for two consecutive time steps to allow the primary agent to infer velocities of obstacles. The agent is rewarded for getting closer to the goal and for reaching the goal. The network is updated according to an update rule, as seen in Algorithm 1, where average_success$_{1,2}$ is the average success during the last 20 episodes for the primary and the adversary agent respectively.

Algorithm 1 Policy update rule

Initialize: $\delta, s_1, s_2 \leftarrow 0$
while training **do**
  $s_1 =$ average_success$_1$
  $s_2 =$ average_success$_2$
  $\delta = s_1 - s_2$
  if $\delta > 0.5$ **then**
    Update adversarial network
  else
    Update primary network
  end if
end while

D. Adversary policy implementations

The following subsection describes the specifications for the adversarial agents policy implementation, and is not concerned with the primary agents policy.

1) Pure Pursuit and LQR Control: The adversarial robot combines a look-ahead pure pursuit strategy with an LQR controller. The pure pursuit part predicts the future position of the primary robot, while the LQR stabilizes the heading error to ensure smooth interception.

For the pure pursuit part, the controller estimates a future position using a short prediction horizon Δt:

$$\hat{x}_0 = x_0 + v_0 \cos(\theta_0)\Delta t,$$
$$\hat{y}_0 = y_0 + v_0 \sin(\theta_0)\Delta t.$$

A pursuit direction is then formed from the adversarial robot to the predicted point.

Algorithm 2 Look-ahead Pure Pursuit

Predict $(\hat{x}_0, \hat{y}_0)$ using the look-ahead time Δt
Compute the vector toward the predicted target
Compute desired heading using $\mathrm{atan2}$

For LQR, heading error is defined as the angle difference between the desired orientation θ_d and the current robot orientation θ_1, normalized to the range $[-\pi, \pi]$:

$$e = \mathrm{atan2}\left(\sin(\theta_d - \theta_1), \cos(\theta_d - \theta_1)\right).$$

The LQR state consists of the heading error and the current angular velocity:

$$x = \begin{bmatrix} e \\ \omega_1 \end{bmatrix}$$

and the steering command is computed using a precomputed gain matrix:

$$a_\omega = -Kx.$$

2) Dynamic acceleration controller: To help the agent learn effectively a curriculum learning strategy was implemented where the difficulty increases as the agent improves. The adversarial agent is controlled by an LQR-based intercept strategy, which predicts the primary agent's future position and steers to intercept it.

Algorithm 3 LQR Steering Control

Compute wrapped heading error
Form state vector with error and angular velocity
Compute steering command using $-Kx$
Apply resulting control

Initially the adversary is stationary ($a_{adv} = 0$), effectively treating it as a stationary obstacle. This allows the primary agent to learn basic navigation and obstacle avoidance. As training proceeds, the linear acceleration of the adversary, and thus the difficulty, is increased. However, the max linear speed is still bounded by `abs(1)` for both agents.

The controller monitors the agent's performance by tracking the outcome of the last 20 episodes. The average success rate is calculated, where a success is defined as reaching the goal without colliding. If the agent's average success rate exceeds 70%, the adversary's linear acceleration is increased ($a_{adv} + 0.05$).

The update logic is described by Algorithm 4.

Algorithm 4 Dynamic acceleration adaptation

Initialize: $a_{adv} \leftarrow 0.0$
while training **do**
  if average_success > 0.7 **then**
    $a_{adv} \leftarrow \min(a_{adv} + 0.05, 1.0)$
    Clear performance history
  else
    Keep current acceleration
  end if
end while

After each acceleration increase the performance history is cleared. This forces the agent to perform consistently on the new, higher difficulty level before the acceleration can be raised further.

3) Sampling-Based MPC with Obstacle Avoidance: The adversarial robot is modeled as a discrete-time unicycle system with timestep Δt:

$$v_{k+1} = \text{clip}(v_k + a_k \Delta t), \tag{1}$$

$$\omega_{k+1} = \text{clip}(\omega_k + \dot{\omega}_k \Delta t), \tag{2}$$

$$\theta_{k+1} = \theta_k + \omega_{k+1} \Delta t, \tag{3}$$

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \begin{bmatrix} \cos\theta_{k+1} \\ \sin\theta_{k+1} \end{bmatrix} v_{k+1} \Delta t, \tag{4}$$

where $\mathbf{p}_k = (x_k, y_k)$ and backward motion is disallowed ($v_k \geq 0$).

A reference trajectory $\{\mathbf{p}_k^{\text{ref}}\}_{k=0}^{N}$ is generated by forward-simulating the primary robot over a fixed horizon $N = 25$ with $\Delta t = 0.2\,\text{s}$.

Obstacle avoidance is enforced as a hard constraint during trajectory rollout. Each obstacle is approximated by an axis-aligned bounding box with safety margin, and a trajectory is infeasible if

$$\left(\frac{x_k - b_x}{a_x}\right)^8 + \left(\frac{y_k - b_y}{a_y}\right)^8 \leq 1, \tag{5}$$

Algorithm 5 Sampling-Based MPC with CEM

Robot states, obstacle set $\mathcal{O}$ Control action u_0
Generate reference trajectory $\mathbf{p}_{0:N}^{\text{ref}}$
Initialize μ, Σ $i = 1$ to N_{CEM}
Sample B control sequences from $\mathcal{N}(\mu, \Sigma)$
Roll out dynamics and evaluate cost J
Discard infeasible trajectories
Update μ, Σ using elite samples
Apply first control input of optimized sequence

at any timestep k. Obstacle avoidance is not included in the reference trajectory.

The MPC objective minimizes tracking error, heading misalignment, control variation, and terminal error:

$$J = \sum_{k=1}^{N} \Big(W_{\text{track}} \|\mathbf{p}_k - \mathbf{p}_k^{\text{ref}}\|^2 + W_{\text{heading}}\, \ell_{\text{heading}} + W_{\text{smooth}} \|\Delta u_k\|^2 \Big) + W_{\text{terminal}} \|\mathbf{p}_N - \mathbf{p}_N^{\text{ref}}\|^2. \tag{6}$$

To estimate the optimal control sequence, a Cross-Entropy Method (CEM) was used. The control sequences

$$u_{1:N} = \{(a_k, \dot{\omega}_k)\}_{k=1}^{N}$$

are sampled from a Gaussian distribution

$$u_{1:N} \sim \mathcal{N}(\mu, \Sigma),$$

with mean $\mu \in \mathbb{R}^{N\times 2}$ and diagonal variance $\Sigma \in \mathbb{R}^{N\times 2}$. Samples are drawn in a normalized space and clipped to $[-1, 1]$.

At each CEM iteration, B samples are rolled out and assigned cost J. Infeasible trajectories receive infinite cost. The elite set $\mathcal{E}$ is formed from the lowest-cost ρB samples, and the distribution is updated as

$$\mu \leftarrow \frac{1}{|\mathcal{E}|} \sum_{u \in \mathcal{E}} u, \tag{7}$$

$$\Sigma \leftarrow \max\big(\text{Var}_{u\in\mathcal{E}}(u), \sigma_{\min}^2\big), \tag{8}$$

where $\sigma_{\min}$ enforces a minimum variance.

The optimized sequence is warm-started at the next timestep by shifting the mean forward in time.

4) Adversarial RL actor: To explore work done in the self-play field of RL, we also tried to implement an adversarial agent who also trained using the TD3 policy optimization algorithm. To this end, we had to reformulate the reward to only consider the distance to the goal position (i.e. the primary agent), for reaching the goal, and for avoiding collisions. The network is updated by the same update rule as the primary agent, seen in Algorithm 1.

V. Results

In this section, the performance of the primary agent when trained under different adversarial conditions is evaluated.

The evaluation is based on three key metrics:

- Success Rate: The percentage of episodes where the agent successfully reaches the goal.
- Collision Rate: The percentage of episodes ending in a collision with an obstacle, a wall or an agent.
- Episode Length: The average number of time steps required to complete an episode (lower is generally better if the task is successful).

The results are presented in Table I, Table II, Table III and Table IV. The first two tables show results from data where controllers trained on 5 million iterations, where table I shows the performance on the *Eval easy* map, which is an easier environment and table II presents the results on the *Eval hard* map, a more challenging environment. The other two tables show results from data where controllers trained on 1 million iterations and table III shows performance on the *Eval easy* map, and table IV shows performance on the *Eval hard* map. The last two tables also include the MPC controller which is absent from the first two.

For the *Eval easy* map, the results are averaged over 100 episodes because the obstacles are placed randomly each time. For the *Eval hard* map, the environment is identical in every run, so the results are based on a single episode.

A. Test Environments

The trained agents were tested in two different environments: one simpler map, and one more difficult map with a new layout.

1) Evaluation Map 1 (Eval easy): Eval easy, shown in Fig. 3, places the primary agent on the left side of a rectangular workspace, requiring it to navigate toward a goal on the right. The map contains one static obstacle and two dynamic obstacles, each designed to introduce distinct navigational challenges.

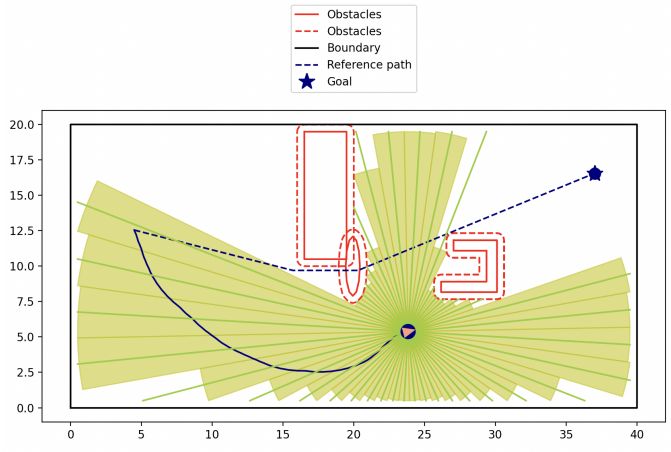


Fig. 3. Evaluation Map 1 (Eval easy) featuring static and dynamic obstacles. The dashed lines represent the reference path, while the solid lines represent the actual path.

2) Evaluate 2 Map (Eval hard): This environment, illustrated in Fig. 4, constructs a curved corridor and evaluates the agent's ability to navigate through a constrained passage populated with both static and dynamic obstacles. The corridor is formed using a sequence of five segments with predefined lengths and nominal turning angles. To prevent excessively sharp turns, each angular increment is clipped using bounds. The resulting polyline is then padded to form left and right corridor walls.

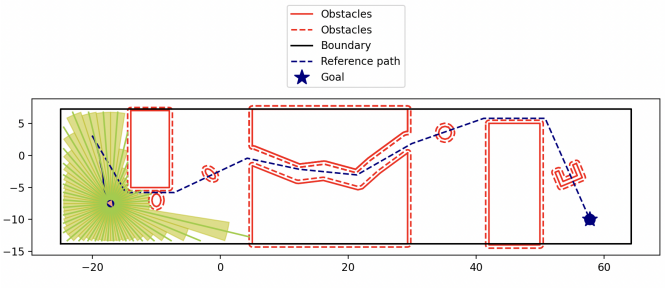


Fig. 4. Evaluation Map 2 (Eval hard) showing the constrained curved corridor. The dashed lines represent the reference path.

TABLE I
EVAL EASY PERFORMANCE 5M (MEAN ± STD OVER EPISODES).

Method	Training map	Success	Collisions	Length
Base Case	Static Obst.	32.0 ± 23.0%	50.1 ± 8.5%	128.1 ± 24.5
No Adversary	Train 1	90.3 ± 5.2%	3.3 ± 2.1%	149.2 ± 7.7
Controller (LQR)	**Adversarial**	**78.8 ± 2.0%**	**14.5 ± 1.8%**	**170.3 ± 2.7**
Controller (LQR)	Adv. Obst.	50.3 ± 2.5%	45.8 ± 2.5%	117.6 ± 2.9
RL Adversary	Adversarial	22.8 ± 2.1%	53.0 ± 2.5%	167.5 ± 4.8
RL Adversary	Adv. Obst.	62.8 ± 2.4%	30.3 ± 2.3%	130.3 ± 2.7

TABLE II
EVAL HARD PERFORMANCE 5M (MEAN ± STD OVER EPISODES).

Method	Training map	Success	Collisions	Length
Base Case	Static Obst.	0.0 ±0.0%	75.0 ±50.0%	112.0 ±126.6
No Adversary	Train 1	0.0 ±0.0%	75.0 ±50.0%	153.0 ±138.2
Controller (LQR)	Adversarial	0.0 ±0.0%	25.0 ±2.2%	231.5 ±5.9
Controller (LQR)	Adv. Obst.	0.0 ±0.0%	100.0 ±0.0%	29.8 ±0.58
RL Adversary	Adversarial	0.0 ±0.0%	100.0 ±0.0%	103.8 ±3.1
RL Adversary	Adv. Obst.	0.0 ±0.0%	100.0 ±0.0%	36.8 ±0.5

TABLE III
EVAL. EASY PERFORMANCE 1M (MEAN ± STD OVER EPISODES).

Method	Training map	Success	Collisions	Length
Base Case	Static Obst.	44.3 ±29.9%	40.0 ±9.5%	129.8 ±26.4
No Adversary	Train 1	91.5 ±4.8%	4.8 ±4.0%	140.2 ±2.0
Controller (LQR)	Adversarial	58.0 ±2.5%	34.0 ±2.4%	168.0 ±3.4
Controller (LQR)	Adv. Obst.	70.5 ±2.3%	20.5 ±2.0%	143.9 ±2.9
Controller (MPC)	Adversarial	54.0 ±2.5%	42.3 ±2.5%	120.4 ±3.2
Controller (MPC)	Adv. Obst.	57.8 ±2.5%	42.3 ±2.5%	102.1 ±1.8
RL Adversary	Adversarial	35.5 ±2.4%	20.3 ±2%	211.2 ±4.4
RL Adversary	**Adv. Obst.**	**87.0 ±1.7%**	**6.8 ±1.3 %**	**149.2 ±2.4**

TABLE IV
EVAL. HARD PERFORMANCE 1M (MEAN ± STD OVER EPISODES).

Method	Training map	Success	Collisions	Length
Base Case	Static Obst.	0.0 ±0.0%	50.0 ±57.7%	171.8 ±148.5
No Adversary	Train 1	0.0 ±0.0%	25.0 ±50.0%	238.5 ±123.0
Controller (LQR)	Adversarial	0.0 ±0.0%	25.0 ±2.2%	231.0 ±6.0
Controller (LQR)	Adv. Obst.	0.0 ±0.0%	0.0 ±0.0%	300.0 ±0.0
Controller (MPC)	Adversarial	0.0 ±0.0%	100.0 ±0.0%	71.5 ±0.2
Controller (MPC)	Adv. Obst.	0.0 ±0.0%	75.0 ±2.2%	116.0 ±5.3
RL Adversary	Adversarial	0.0 ±0.0%	100.0 ±0.0%	56.3 ±0.45
RL Adversary	Adv. Obst.	0.0 ±0.0%	75.0 ±2.2%	186.8 ±3.7

VI. DISCUSSION

As the MPC controller was unable to complete five million training iterations, two separate tables are presented using results from one million iterations to enable a consistent comparison across all controllers. This led to several noteworthy observations; for example, the reinforcement learning agent trained on the adversarial obstacles map, achieved a higher success rate after one million iterations (87%) than after five million iterations (62.8%), a trend that was also observed in the baseline configuration.

While the base case primarily serves as a comparative benchmark, the results nonetheless yield several noteworthy insights. First, the collision rate on the Eval. Hard map was 100% across all trials, with the exception of a single run in which the agent was trained on both maps. In this case, the agent survived until the maximum episode length without either making progress or colliding. This abnormal behavior accounts for the increased average episode length observed in that setting, as well as the corresponding reduction of the collision rate to 75%. Second, the agent trained exclusively on static obstacles exhibited substantially poorer performance compared to the agent trained on the Train 1 map, which includes both static and dynamic obstacles. This performance disparity is expected, as the evaluation maps contain both obstacle types and therefore favor agents exposed to a more diverse training environment. Finally, with respect to the number of training iterations, the primary agent demonstrated improved performance after one million episodes compared to five million episodes for both training configurations. One plausible explanation for this trend is overfitting to specific map layouts after excessive training, which may cause the navigation policy to adapt too closely to the training scenarios rather than learning more generalizable obstacle-avoidance.

The LQR controller trained with the adversarial map, for 5 million iterations, achieves a success rate of approximately 70%, with a collision rate of 24.5%, see Table I. This represents a significant improvement over the base case and the RL adversary. However, when trained on the adversarial obstacles map for 5 million iterations, the performance degrades notably, with a lower success rate of 50.7% and increased collision frequency of 44%. This indicates that the dual challenge of evading a skilled adversary while navigating static clutter may have been too complex during training, preventing the agent from converging to an effective policy.

The agent trained against the LQR controller also has a long average episode length. Since the success rate is high, this means the agent is navigating carefully rather than just crashing early. Training against a predictable adversary taught the agent to be safe but slow. However, the results after 1 million iterations show an inverted performance pattern for the LQR controller, as presented in Table III. The adversarial map achieves a success rate of 58% with a collision rate of 34%, whereas the adversarial obstacle map attains a higher success rate of 70.5% and a lower collision rate of 20.5%. The underlying reason for this behavior is difficult to determine, as it may be influenced by multiple factors. Additional training runs would be required to identify the exact cause.

Evaluation results on the Eval easy map illustrate the impact of training against a sophisticated MPC adversary compared to the base case (44.3% success, 40% collisions). Training with an MPC adversary, characterized by a long prediction horizon ($N = 25$, 5s) and aggressive tracking weights ($W_{\text{track}} = 10$, $W_{\text{terminal}} = 30$), appears to produce a more cautious policy. While this training regime improves success rates to 54–57.8%, it also leads to higher collision rates (42.3%) and shorter episode lengths (102–120 steps), suggesting the agent learns to prioritize direct, high-risk paths when its training environment is highly constrained. This effect is further amplified on Eval hard. In these narrow, cluttered corridors, the agent fails to generalize effectively, resulting in 0% success and 75–100% collision rates. The highly controlled and predictable motion of the training adversary (enforced by $W_{\text{heading}} = 2$ and $W_{\text{smooth}} = 0.1$) likely causes the agent to overfit to specific blocking maneuvers, leaving it unable to navigate the dense static obstacles of the hard evaluation set. Ultimately, while MPC-based training can improve performance on simpler maps, the resulting policy remains highly sensitive to map geometry.

The RL agent showed slightly better results, but differed between model versions. For the model trained over 5 million steps, the Adversarial map resulted in poor performance (25.0% success), compared to the 67.0% for the adversarial obstacles map. This trend continues with the results after only training for 1 million steps, where the agent performed almost on par with the *No Adversary* model. There are many possible explanations for this reduction in performance, but one of the most plausible explanations is due to an implementation level defect which caused the agent to agent collision to be offset by one agent radius. This explanation is also consistent with the poor results from only training with an adversary agent, since then the only collisions available to the RL agent would suggest that this is the true collision distance, which is not true. On the Eval hard map, all agents fail completely. The success rate drops to zero and the collision rate is 75%–100% (Table II).

VII. Conclusion and Further Work

The findings of this study provide partial support for the central hypothesis. Training the primary agent against an adversarial controller, specifically the LQR controller, resulted in a substantial improvement in robustness to dynamic obstacles compared to the static baseline, as evidenced by a higher success rate (78.8% versus 32.0%). This outcome supports the first contribution of this work, namely that incorporating adversarial agents during training can enhance the robustness of navigation policies in dynamic environments.

Despite these improvements, to further improve the results of the LQR controller, additional investigation of the success rate is justified. Adjusting the success criteria based on episode duration may provide a more informative evaluation. For longer runs, stricter success criteria may be appropriate, whereas shorter episodes may require a more tolerant evaluation to avoid penalizing otherwise stable behavior. However, the results also highlight critical limitations. The poor performance of the RL adversary in open environments suggests that simply introducing a learning opponent is not enough. The training environment itself must be sufficiently complex (e.g., including static obstacles) to encourage the agent to learn robust policies. Furthermore, the universal failure on the Eval hard map demonstrates that while adversarial training improves robustness within familiar maps, it does not automatically provide the ability to generalize to completely new environments.

Since the success rate is 0% across all cases on the Eval hard map, this clearly indicates that the map is too complex for the agent to succeed. Consequently, a suggestion for future work is to reduce the complexity of the map, for example, by widening the corridors so that it remains more challenging than Eval easy but less difficult than Eval hard. In this way, the results presented in Table II and IV would likely provide more meaningful insight and add greater value to the study.

Several additional limitations were identified during this project. In particular, two implementation-level defects were discovered: one related to the vector transformation process and another concerning the computation of distances between the primary agent and surrounding objects. Addressing these issues is expected to improve the fidelity and robustness of the system and should therefore be prioritized in future work.

Finally, since this study focused on implementing and comparing different methods rather than extensive parameter optimization, future work could investigate more systematic tuning of the reward structure. In particular, the LQR controller, which showed the strongest performance after 5 million iterations, is well suited for further improvement through controller refinements and additional hyperparameter tuning to enhance performance and robustness. Overall, these findings suggest that adversarial training can be beneficial for improving policy robustness; however, further investigation is needed to determine how such robustness scales when applied to already well-performing environments, such as the Train 1 map, and under more diverse evaluation conditions.

References

[1] P. Nguyen, T.-H. Wang, Z.-W. Hong, E. Aasi, A. Silva, G. Rosman, S. Karaman, and D. Rus, "Regen: Generative robot simulation via inverse design," 2025. [Online]. Available: https://arxiv.org/abs/2511.04769

[2] S. Xiang. (2025) Basic pure pursuit. purduesigbots. [Online]. Available: https://wiki.purduesigbots.com/software/control-algorithms/basic-pure-pursuite

[3] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent auto-curricula," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[4] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning*, 2014, pp. 387–395.

[5] 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," 2019. [Online]. Available: https://arxiv.org/abs/1509.02971

[6] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[7] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, 1989.

[8] R. Bellman, *Dynamic Programming*. Princeton University Press, 1957.

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

[10] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[11] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*. PMLR, 2018, pp. 1587–1596.

[12] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Technical Report CMU-RI-TR-92-01, January 1992. [Online]. Available: https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf