Missile Guidance

FINAL PROJECT

Machine Learning and Neural Network Design

YOGEV ATIAS || YEHONATAN DAHAN

Table of Contents

List of figures

The task

Our work focuses on the control of guiding a 2-dimensional free-falling missile in order to hit a target. This task involves making real-time decisions to apply horizontal forces along the trajectory to achieve the desired outcome. In this project, the focus is on applying deep reinforcement learning (DRL) to solve the task. Each episode / scenario is characterized by the missile's initial conditions and a targets location. The actual trajectory is an outcome of the systems dynamics and the agent's actions, which will be further discussed in this writeup. To schematically illustrate our task, the following figure is presented:

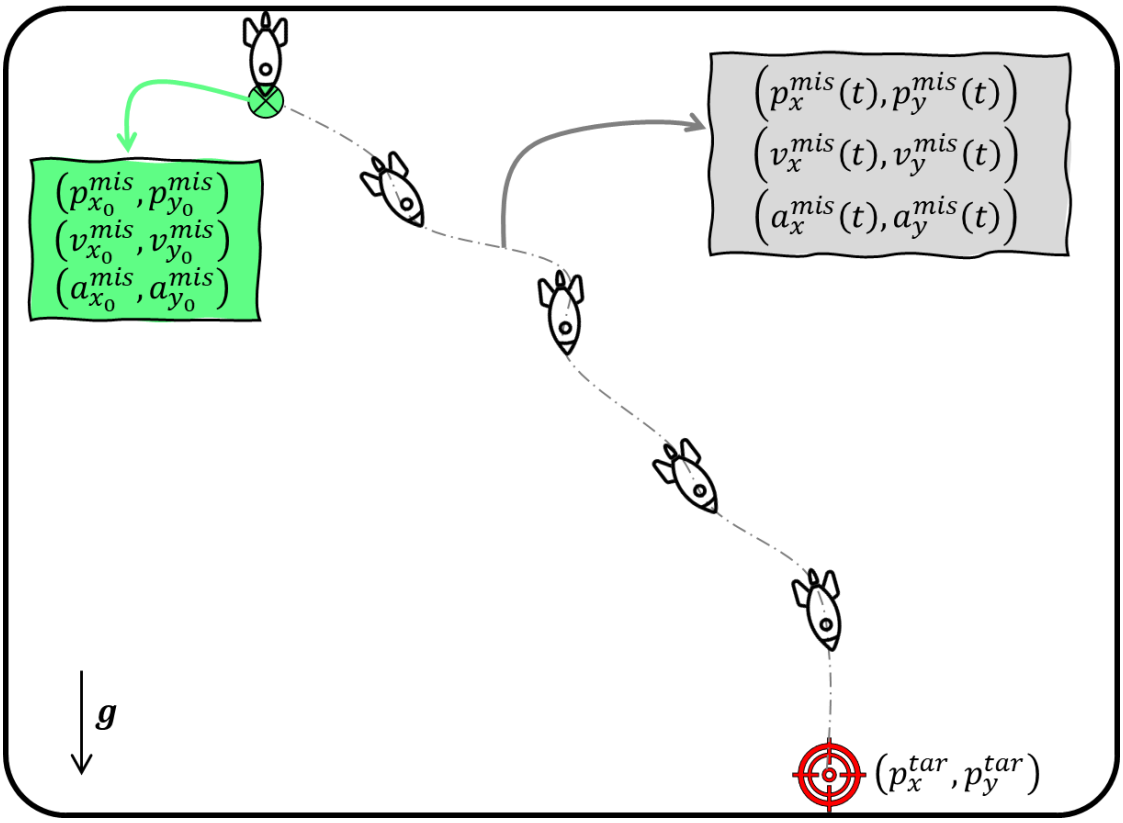

Figure 1 - Objective's illustartion

In Figure 1, the missile's and the target's states are denoted as (mis) and (tar) respectively, the position, velocity and acceleration are denoted as p, v and a. Furthermore, the elements along each axis are marked with x and y, and the initial and the time dependent states are denoted as "0" and (t) respectively. Moreover, as can be seen in Figure 1 - a global / primary co-ordinate system was not defined. This is due to the fact that only the relative positions between the missile and the target matter, as will be explained later in this writeup.

State and Action Spaces

The state space (SS) is a closed form equation which holds our system's dynamics (state transition) and observation models:

State transition model

In the context of missile guidance, we are concerned with the tracking missile's position (both horizonal and altitude). For that purpose, data such as the missile's current position, velocity, altitude are all needed and presented with the discrete dynamic equation:

$$\begin{bmatrix} p_{x,t+1}^{mis} \\ p_{y,t+1}^{mis} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & 0.5 \cdot \Delta t^2 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & 0.5 \cdot \Delta t^2 \end{bmatrix} \cdot \left[p_{x,t}^{mis}, p_{y,t}^{mis}, v_{x,t}^{mis}, v_{y,t}^{mis}, a_{x,t}^{mis}, a_{y,t}^{mis} \right]^{\top}$$

Where $t \in \mathbb{N}$ is a discrete time-step. In our case, the initial conditions are:

$$p_{y0}^{mis} = 100\ [m]\ , p_{x0}^{mis} = 50\ [m]\ , v_{x0}^{mis} = v_{y0}^{mis} = 0\ \left[\frac{m}{s}\right]\ , a_{y}^{mis} = -g = -9.81\ \left[\frac{m}{s^2}\right]$$

Observations

The observation space represents the set of all relevant information and data that the agent can perceive or sense at a particular time step. The observation serves as the agent "sensory input" and provides crucial information for decision making. The agent observation in our case:

$$y_t = \begin{bmatrix} p_{x,t}^{mis} \\ p_{y,t}^{mis} \\ p_{x,t}^{tar} \\ p_{y,t}^{tar} \end{bmatrix}$$

Denote that the agent observes the targets y position, although it is constant 0. This is used to generalize the expression of the reward function which will be further explained.

Action space

The action space defines the set of actions the RL agent can take in response to its current state. In your project, the agent's actions consist of applying horizontal forces to the gliding missile. The agent needs to determine the magnitude and direction of the force to influence the missile's trajectory effectively. The action is a_x^{mis}.

State Space equations

One may formalize the state transition model and the observations, into a linear, state space model in the form:

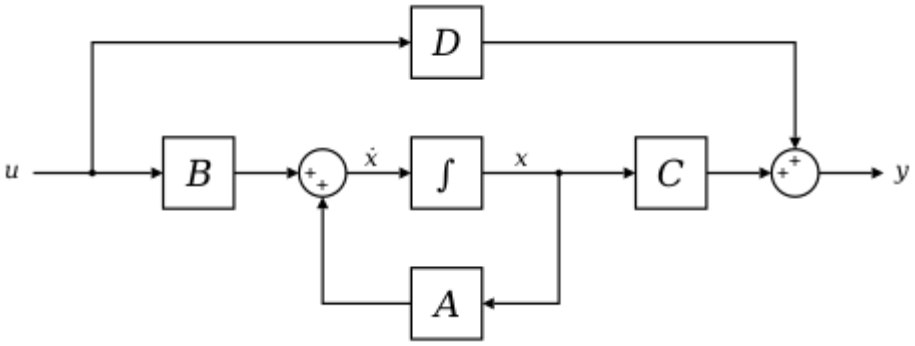

Figure 2 - Linear SS (taken from Wikipedia)

Discretizing the continuous equations leads to:

$$\bar{x}_{t+1} = \bar{\bar{A}}_t \cdot \bar{x}_t + \bar{\bar{B}}_t \cdot \bar{u}_t$$

$$\bar{y}_t = \bar{\bar{C}}_t \cdot \bar{x}_t + \bar{\bar{D}}_t \bar{u}_t$$

Where $\bar{\bar{A}}_t, \bar{\bar{B}}_t, \bar{\bar{C}}_t$ and $\bar{\bar{D}}_t$ are independent of time, thus their "t" denotation will be left out and $\bar{u}_t$ is a scalar, thus its "bar" mark will also not be written. These notation, lead to the definitions:

$$\bar{x}_t = \left[p_{x,t}^{mis}, p_{y,t}^{mis}, v_{x,t}^{mis}, v_{y,t}^{mis}, a_{x,t}^{mis}, a_{y,t}^{mis}, p_{x,t}^{tar}, p_{y,t}^{tar}\right]^{\top}$$

$$\bar{\bar{A}} = \begin{bmatrix} 1 & 0 & \Delta t & 0 & \Delta t^2 & 0 & 0 & 0 \\ 0 & 1 & 0 & \Delta t & 0 & \Delta t^2 & 0 & 0 \\ 0 & 0 & 1 & 0 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} ; \; u_t = f_{x,t} \; ; \; \bar{\bar{B}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1/m^{mis} \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\bar{\bar{C}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} ; \; \bar{\bar{D}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Where m^{mis} is the mass of the missile.

Agent

An agent refers to an autonomous entity or program that interacts with an environment in order to achieve specific goals. Agents are a fundamental concept in the field of RL and are used to model systems that can make decisions and take actions to maximize cumulative rewards or achieve objectives. In our work, we have chosen a "DDPG" agent as our agent. Before diving into this specific agent and why we chose it, we first explain some important terms:

Deterministic Policy Gradient (DPG)

As learned in class, the policy function could be treated as a probability function or a distribution over the possible actions. This means that for some state $\bar{x}_t$ (belonging to all the possible states in state space), each possible action from the action space is getting a probability. In deterministic policy gradient approach, instead of matching a distribution over the possible actions, each action gets a deterministic value (as its name suggest), usually denoted by the function $\mu(\cdot)$. In other words, for some state $\bar{x}_t$ from the state space, a specific action a_t exists, namely: $a_t = \mu(\bar{x}_t)$. In order to perform the optimization, a gradient of the loss function must be defined (in order to maximize the reward) – which was suggested and proved by the paper Deterministic Policy Gradient Algorithms (Theorem 1):

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s\sim\rho^\mu}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)\,\middle|_{a=\mu_\theta(s)}\right]$$

Where s is a state, a is an action, $J(\cdot)$ is the performance objective, defined as:

$$J(\mu_\theta) = \mathbb{E}\left[r_1^\gamma \,\middle|\, \mu\right] = \mathbb{E}_{s\sim\rho^\mu}[r(s,a)] = \mathbb{E}_{s\sim\rho^\mu}\left[r\big(s,\mu_\theta(s)\big)\right]$$

Where $r_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t}\, r(s_k, a_k)$ is the total discounted award (and $0 < \gamma < 1$ is the discount factor), $r(\cdot)$ is a specific reward and $\rho^\mu(s)$ is the discounted state distribution. The action-value function, denoted as Q^μ, represents the expected cumulative reward that can be obtained by following a particular policy. Computing this function isn't simple, and some methods are Monte-Carlo (MC) evaluation or temporal-difference learning.

Q-learning

As mentioned, learning the action-value function Q^μ or $Q(s,a)$ is a very complicated (but important) task. The famous term "Q-learning" refers to the algorithm that distinguishes the value of taking some acting in a current state, without relying on the model or the environment. By calculating the rewards of future states, the algorithm is able to find an optimal policy, which leads to a maximum reward. The update term of Q-learning algorithm is as follows:

$$Q^{new}(s_t,a_t) = (1-\alpha)\cdot \underbrace{Q(s_t,a_t)}_{Current\ value} + \alpha \cdot \underbrace{\left(r_t + \gamma \cdot \max_a Q(s_{t+1},a_t)\right)}_{New\ value}$$

As can be seen, the algorithm is using a weighted sum (using the learning rate α) between the current value and the new value, defined with the reward and the "discounted" action maximizing the next state's s_{t+1} Q-value.

Deep Deterministic Policy Gradient (DDPG)

The DDPG (Deep Deterministic Policy Gradient) algorithm is a reinforcement learning (RL) technique used for solving continuous action space problems. It is an extension of the DPG (Deterministic Policy Gradient) algorithm that combines elements of deep neural networks and the actor-critic architecture. DDPG is particularly useful for tasks where the action space is continuous. The DDPG general architecture is as shown below:

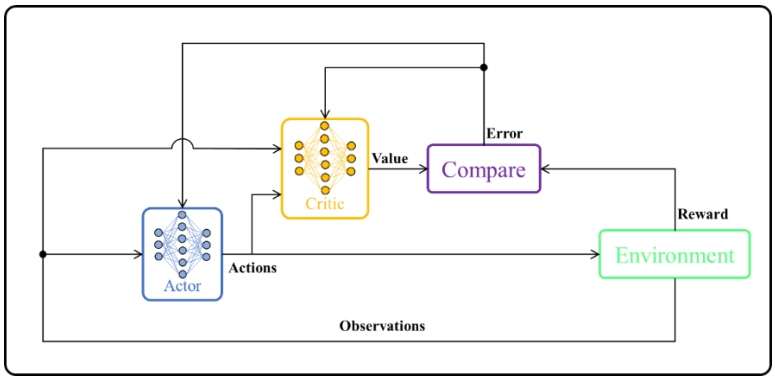

Figure 3 - DDPG Architecture

The DDPG algorithm, is as follows (from OpenAI's website):

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_\phi(s, a) - y(r, s', d))^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Figure 4 – DDPG algorithm Pseudocode

Reward Function

To teach the RL agent, we have designed a reward system that provides feedback on its actions. The rewards are crucial for the agent to learn. We defined the reward as follows:

$$R(t) = \begin{cases} -\sqrt{(\Delta p_{x,t})^2 + (\Delta p_{y,t})^2} & p_{y,t}^{mis} > 0 \\ \begin{cases} +1000 & \Delta p_{x,t} < 3 \\ +500 & \Delta p_{x,t} < 15 \\ +100 & \Delta p_{x,t} < 25 \end{cases} & p_{y,t}^{mis} = 0 \end{cases}$$

The reward consists of 2 parts:

Immediate Reward (Penalty)

A negative reward (in other words, a penalty) is constantly given to the agent, with an amount proportional to its Euclidian distance from the target and can be seen in Figure 5. This way, the agent learns to optimize its actions to maximize its cumulative reward over time. In other words, the agent tries to minimize the penalty along the free fall by applying force to get closer and closer to the target (as much as possible):

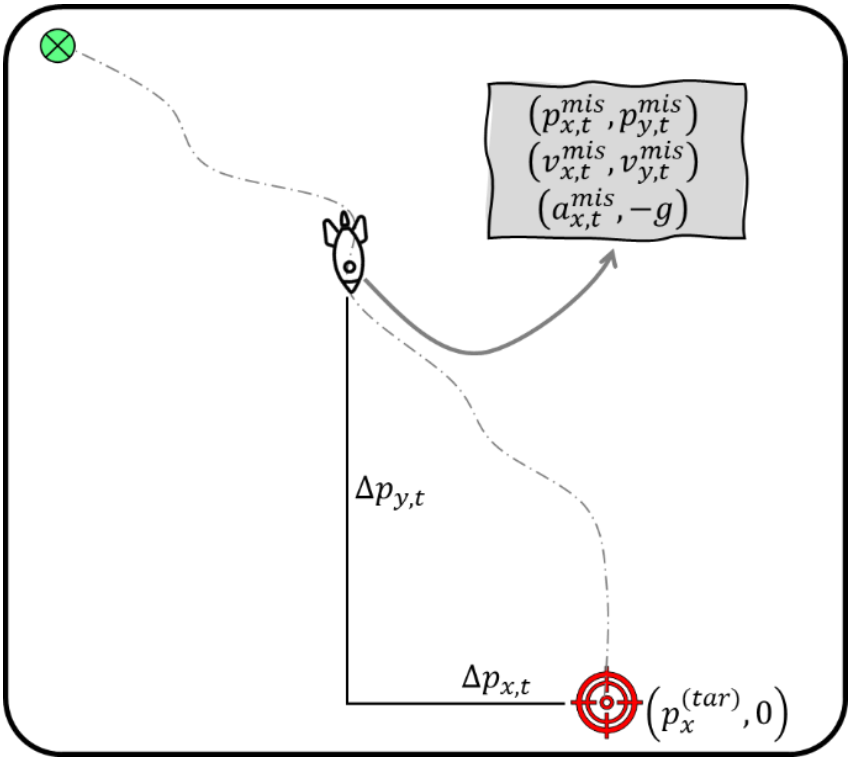

Figure 5 - Reward illustration

Denote that $\Delta p_{x,t}$ and $\Delta p_{y,t}$ are taken with their absolute values and thus positive by definition.

Final Reward (Bonus)

A positive reward is granted at the end of each scenario according to the hitting distance from the target's location. As can be seen (and explain before), the closer to the target the missile hits, the higher reward it gets for the episode, as demonstrated below:

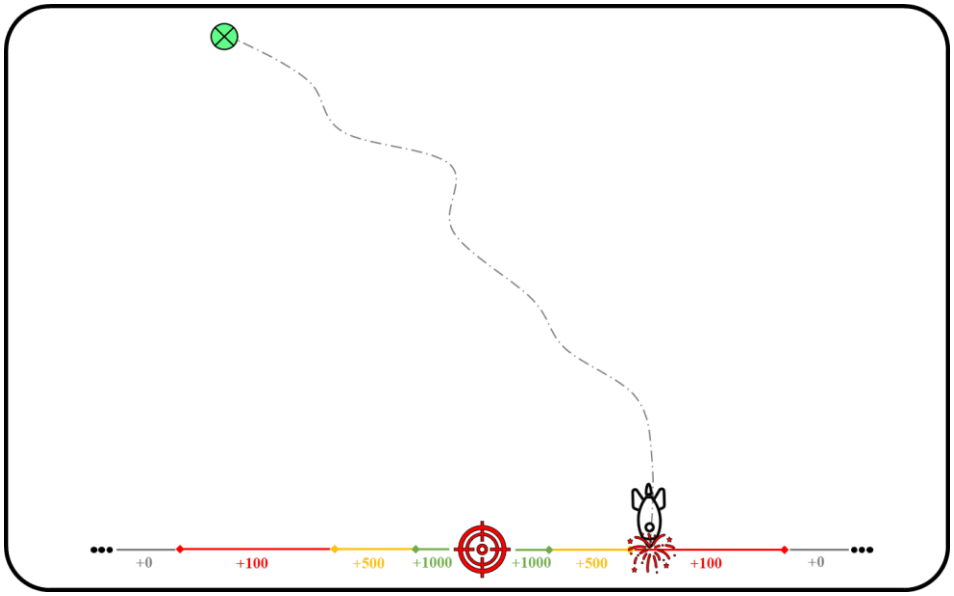

Figure 6 – Hitting location reward

Thus, by hitting close to the target, its able to minimize the penalty summed along the trajectory.

Denote, that the Reward equation presented above, is a function of the **difference** (Δ) between the target and the missile, leading to a symmetry between training the agent with varying target locations as shown in Figure 7 and training the agents with varying initial position as shown in Figure 8:

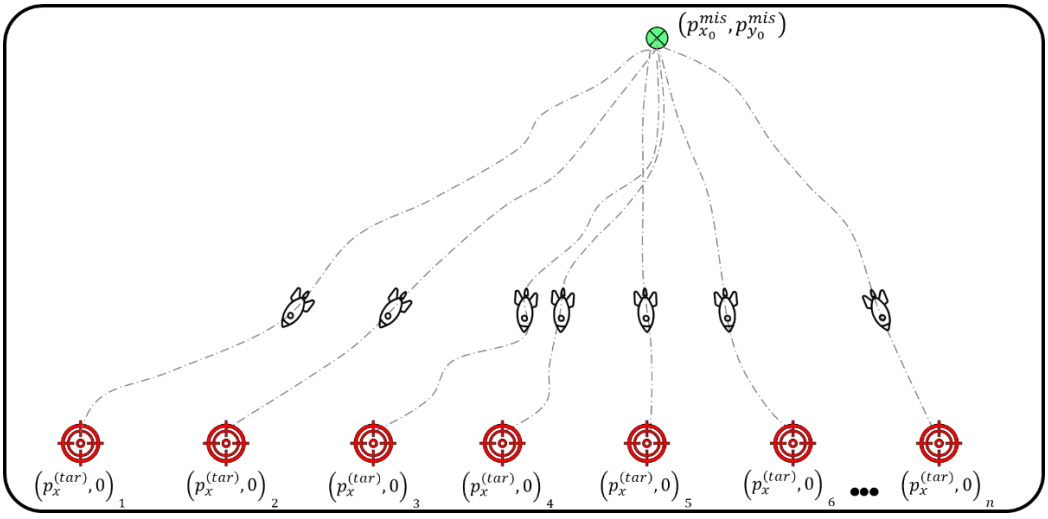

Figure 7 - Learning trajectories with varying target locations

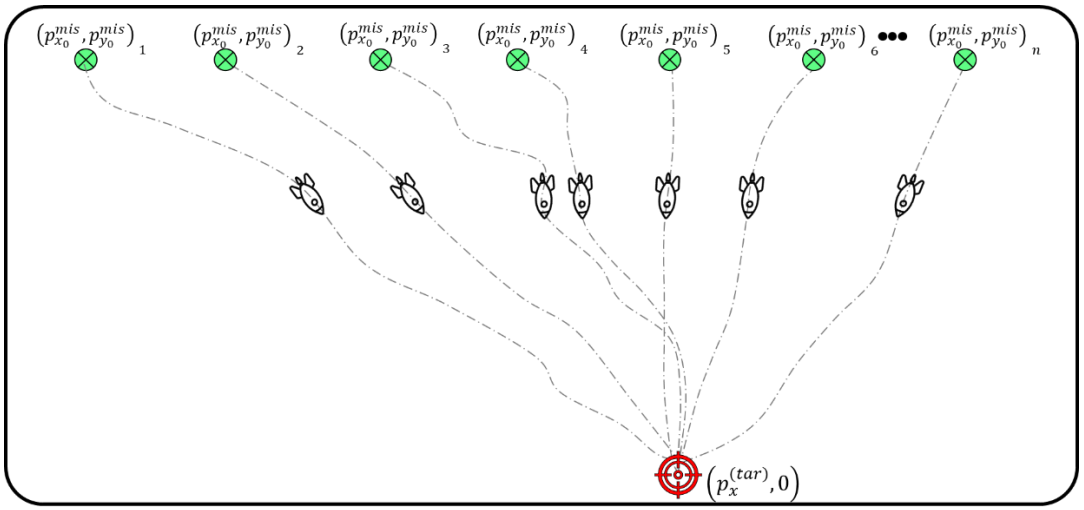

Figure 8 - Learning trajectories with varying missile's initial position

This symmetry is an outcome of the fact that the absolute position of the target and the missile does not really matter and is not included in the reward.

Results

This chapter discusses the results of our work:

Mean absolute error

To emphasize the symmetry described above, a comparison between the two approaches (varying initial position, and varying target location) is shown:

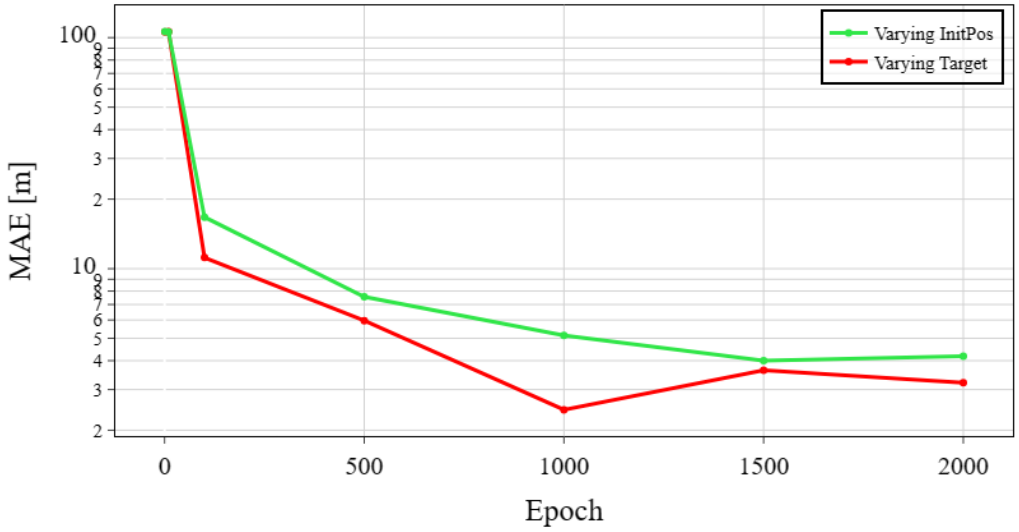

Figure 9 - Comparing between the two approaches (static target)

In the figure above, the mean absolute error (MAE) is representing the average error of the missile's hitting location in meters:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|p_x^{mis}\left(p_y^{miss}=0\right)-p_x^{tar}\right|$$

Where n=100 is the number of testing scenarios. As can be seen, the agent was able to handle (and converge to an error <10[m]) cases of varying initial condition, with constant target location, although it was trained in an opposite manner – a constant initial position with varying target location.

In Figure 9, the Initial position was randomly drawn from a uniform distribution, such that $\left(p_x^{mis},p_y^{mis}\right)_i \sim U(40,60)$. The same hold for the scenario where the target's location was randomly picked, such that: $(p_x^{tar})_i \sim U(40,60)$. To demonstrate the generalization of our agent, the figure below shows the mixed scenario – both varying initial position and target location, each picked from the uniform distribution described above:

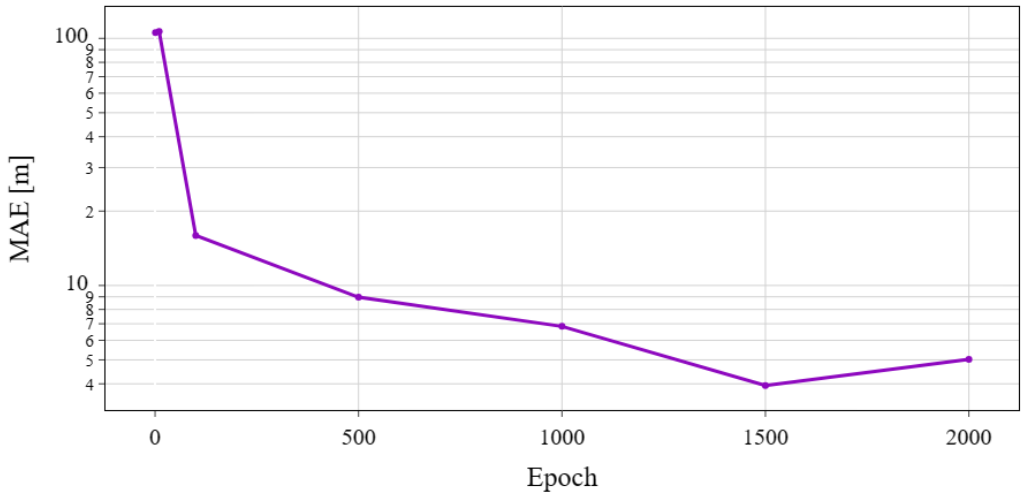

Figure 10 - Mixing the two approaches (static target)

As can be seen, even when randomizing both the missile's initial position and the target's location – the agent is able to converge to a mean absolute error <10[m] after 500 epochs.

In order to challenge the agent, we also tested it on a more difficult scenarios, where both the missile's initial position and the targets' locations are picked from a wider range of distribution: $\left(p_{x0}^{mis},p_{y0}^{mis}\right)_i, (p_x^{tar})_i \sim U(10,90)$:

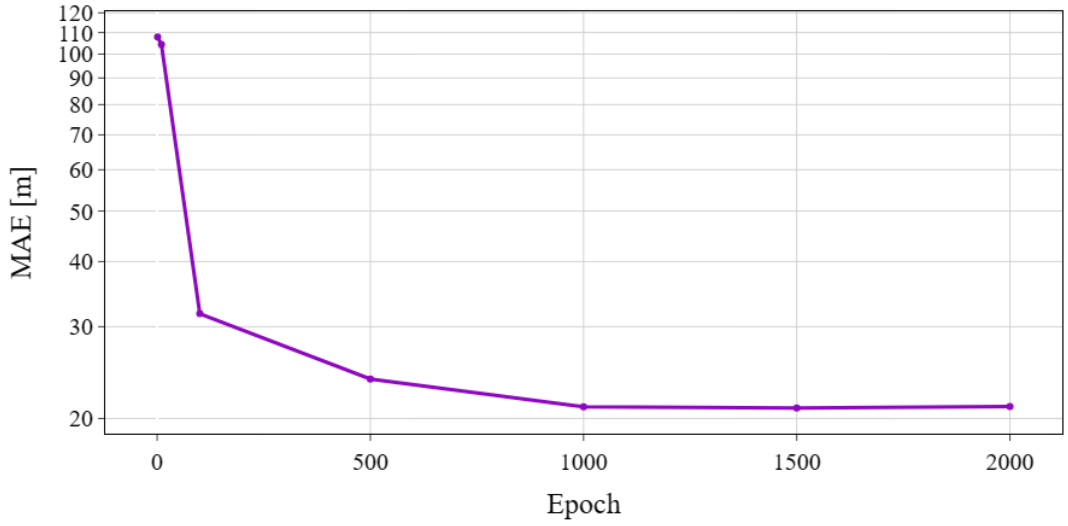

Figure 11 - Mixed approaches with wider range

As can be seen, even when tested on scenarios much harder than the training episodes, the agent managed to achieve reasonable results: ~20[m] mean absolute error.

Learning Progression

To visualize the learning progression, the following figure shows the results of the different stages of the learning progression - versus a constant target location, i.e.: the agent learned using varying locations, and the different learning stages is demonstrated versus a constant, specific target to show in the progress:

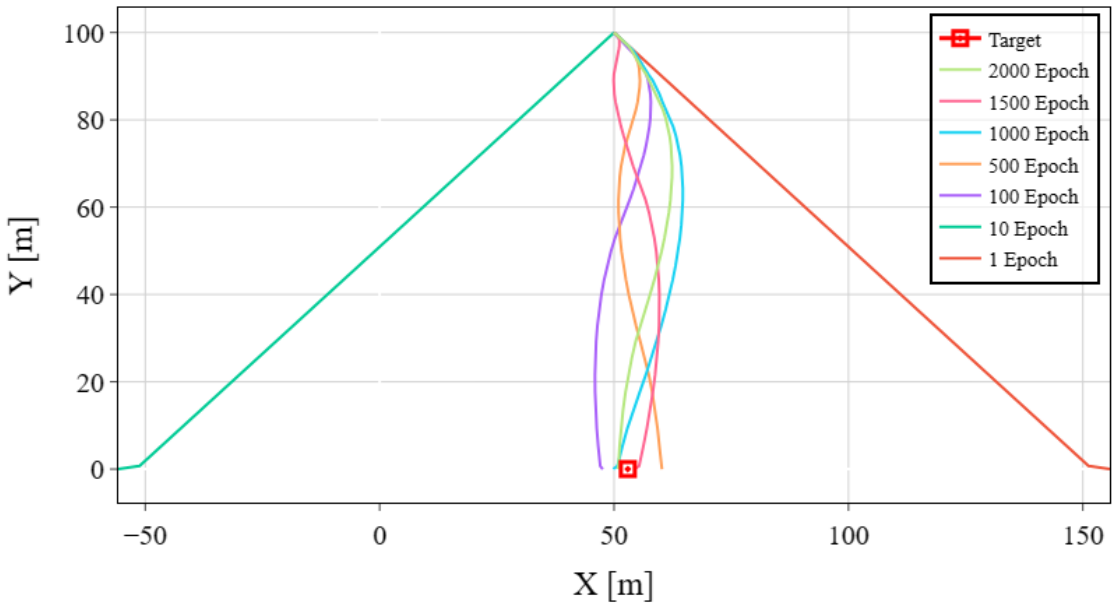

Figure 12 - Trajectories for different learning stages

As can be seen, the agent initially failed to hit the target's area, until around the 100th epoch, where then it started hitting very close to the target.

Moving Target

To test our agent on a more difficult scenario, it was re-trained to a new task – a dynamic target that changes its position along the x-axis with time, as demonstrated in the following figure:

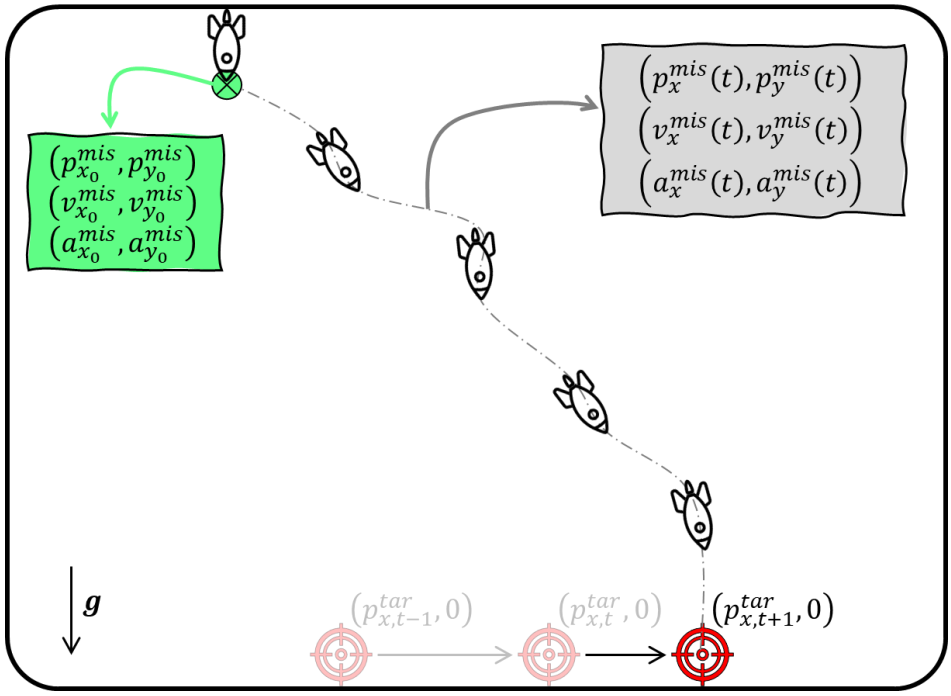

Figure 13 - Moving tarrget illustration

As can be seen, the agent has to predict the target's future location in order to hit it.

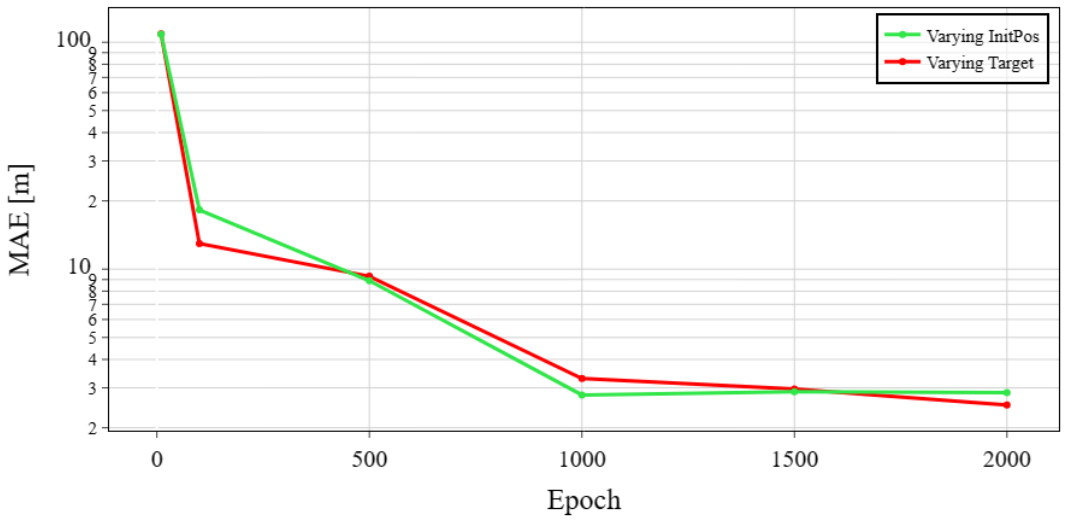

Figure 14 - Comparing between the two approaches (moving target)

As can be seen in Figure 14, the agent successfully converged to a very small mean absolute error of ~3[m] with both approaches: both randomizing the initial position of the missile and randomizing the initial location of the moving target.

Just as done with the static target, we examined (without re-training the agent) the harder scenario where both the missile initial position and the target's initial location are randomly picked:

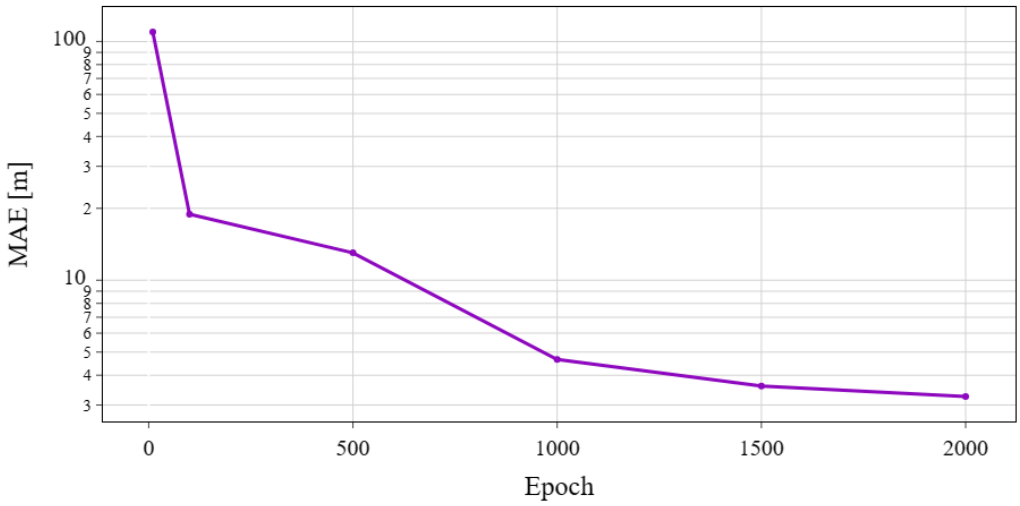

Figure 15 - Mixing the two approaches (moving target)

Figure 15 shoes that the agents succeeded with a really small mean absolute error (~3[m]) even for this scenario, as also resulted in the static target example.

Conclusion

In conclusion, our project delved into the challenging task of guiding a free-falling missile to hit a target using deep reinforcement learning (DRL). We developed a comprehensive understanding of the state and action spaces, state transition models, and observation systems necessary for this task. We implemented the DDPG (Deep Deterministic Policy Gradient) agent to solve this problem, showcasing its effectiveness in optimizing missile trajectories. Our reward function, which included both immediate penalties and final rewards based on hitting location, motivated the agent to minimize errors and approach the target accurately. We demonstrated the agent's ability to generalize its learning across varying initial positions and target locations, showcasing robust performance even in scenarios it was not explicitly trained for. Additionally, we examined the agent's adaptability to a moving target, where it had to predict the target's future location, and found that it successfully adapted to this more challenging scenario.

In summary, our project highlights the power of deep reinforcement learning in solving complex real-time control tasks. The DDPG agent showed strong learning capabilities, robustness to changing conditions, and the ability to adapt to dynamic environments. This work could contribute to the growing field of autonomous control systems and demonstrates the potential of DRL in various practical applications.