Implementation of Model-based and Model-free strategies for Multi-player Pursuit-Evasion Games

Report submitted in the partial fulfilment of the requirements for B.Tech. Project

by

Nikhil S

EE20B090

Email : ee20b090@smail.iitm.ac.in

Under the Supervision of

Prof. Bharath Bhikkaji

Department of Electrical Engineering

Indian Institute of Technology Madras

Acknowledgement

On the submission of my thesis titled "**Implementation of Model-based and Model-free strategies for Multi-player Pursuit-Evasion Games**", I would like to take this opportunity to express my sincere gratitude towards my supervisor Prof. Bharath Bhikkaji for his invaluable guidance and support. His patience, the freedom he provided me during the course of the project, enabled me to conduct research in an effective manner. I thank all the faculties of Department of Electrical Engineering, the Indian Institute of Technology Madras for the valuable education and training imparted that made my under-graduation fruitful and productive. I also thank Mr. Abinash Agasti for his extensive support and guidance during the entire course of the project, therby making the entire process very smooth. I express my heartiest affection to all my inspiring seniors who have guided and motivated me from the beginning. Their inspiring guidance has pushed me to achieve my full potential through out my under-graduation. I thank my friends and batch-mates for the memorable journey of last four years. Last but never the least, my parents who have always stood up as a beacon of light all through my life. Words cannot rightly sum up the gratitude that I feel towards their never-ending support.

Contents

1 Introduction

Multi-Agent Pursuit Evasion Games (PEGs) are a class of differential zero-sum games involving pursuers whose goal is to capture the evaders and evaders whose goal is to avoid the pursuers. These games represent a complex and challenging domain within the field of game theory and multi-agent systems. PEGs find applications in various domains, including security, surveillance, robotics, and military operations, where the strategic interaction between pursuers and evaders is of paramount importance.

A particular variant of this game is the Target Guarding Problems (TGP), where a team of pursuers seeks to defend the target by capturing a team of evaders whose aim is to reach the target while evading the pursuers. These have been extensively studied in the literature in various forms in the context of optimal control and differential games; however, single pursuer-multiple evader games have not been explored in detail. Further, as the complexity of the game increases, finding optimal solutions analytically is a very hard problem, and there is a requirement to find optimal strategies without knowing the models of the agents in the game. To this end, this thesis explores and delves into the implementation of model-based and model-free strategies for different forms of pursuit-evasion games.

1.1 Literature Review

Differential Game theory has been studied for decades and as early as 1965 by Isaac[10], where he employed the principles of game theory, calculus of variations, and control theory, to solve problems involving a dynamic conflict between multiple agents/players. Later, Bellman known for the method of dynamic programming, provided a tool whereby state feed- back optimal strategies could be directly obtained as opposed to method of characteristics by Isaac [4]. Further, with the work of Pontryagin, differential game theory was formulated and was formally studied. The pursuit-evasion games have further been studied to synthesize saddle-point strategies that provide guaranteed performance for each team regardless of the actual strategies implemented by the adversary. This has been studied in various settings with different number of pursuers and evaders, with 1v1

[5], nv1 [14] [9] and the general nvm [12][7]. These works usually involve modelling the individual players and finding the optimal play in closed form, using optimal control techniques. The later works involve finding solutions to the Hamilton-Jacobi-Isaac (HJI) Partial Differential Equation (PDE) by guessing an appropriate value function. However, the 1vn case is rarely explored with some works by [6][13] where the pursuers are considered to be faster and other advantages provided to the pursuer. There is no work on homogeneous 1vn game which also analyses the similarities between 1v1 and 1vn games for evaders. This thesis address these arenas providing hardware implementation along with it.

While Model-based techniques provide a structure to the problem, the complexity in solving the HJI-PDE for complex environments increases tremendously. With the advent of Reinforcement Learning (RL), Multi-Agent Reinforcement Learning (MARL) has evolved and several research work has been done on coordinating multi-agents to solve a given task[3]. To address the challenges associated with coordination and collaboration in MARL, researchers have developed various algorithms and frameworks. One prominent approach is centralized training with decentralized execution (CTDE), where agents share information during training to learn a centralized policy, but execute decentralized policies during interaction with the environment. Another notable direction in MARL is multi-agent actor-critic methods, which extend the actor-critic architecture to multi-agent settings. These methods leverage centralized critics to estimate the value functions of joint actions, enabling agents to learn coordinated policies through decentralized execution. Variants of multi-agent actor-critic algorithms, such as MADDPG (Multi-Agent Deep Deterministic Policy Gradient)[11] and COMA (Counterfactual Multi-Agent Policy Gradients[8]), have demonstrated success in challenging coordination tasks, including cooperative navigation, competitive games, and communication-based tasks. In this thesis, 1v1 version of TPG is explored using an on-policy MARL algorithm, MAPPO(Multi-Agent Proximal Policy Optimisation)[15] owing to its computational simplicity and data efficiency.

1.2 Thesis outline

This thesis discusses model-based and model-free strategies for solving multi-player pursuit-evasion games.

- In the first section, the approach towards solving a single-pursuer, multiple evaders game analytically is discussed, providing optimal strategies to all the agents. Further, a comparative analysis is done on various objective functions that aid in formulating the optimal strategy for the pursuer. This is followed by a discussion on the hardware implementation of a 1v2 game.
- In the second section, MARL definitions and techniques in solving this competitive game are discussed, followed by the implementation details in a simulation environment of an off-the shelf MARL algorithm (MAPPO), followed by some preliminary results and analysis.

Overall, this thesis aims to contribute to understanding optimal strategies for multi-player pursuit-evasion games and provide insights into the strengths and limitations of model-based and model-free approaches in this domain.

2 Single-Pursuer Multi-Evader Games

This chapter discusses the problem formulation and analytical solutions of the single-pursuer, multi-evader game, where all the agents have uniform and equal speeds, followed by the hardware implementation details.

2.1 Problem Setting

A Target Guarding Problem (TGP) with a single pursuer and n evaders is considered. The pursuer aims to capture these evaders before they reach the target, while the evaders aim to reach the target while avoiding the pursuer. The agents are denoted by $E_i, i \in \{1, 2, ...n\}$ for evaders and P for the pursuer and are considered to be holonomic and two-dimensional with their state of the form $\mathbf{x} := (x, y)$. This state also represents their position in this two-dimensional space. The dynamics of the agents are given by:

$$\begin{aligned} \dot{x}_i(t) = \cos\psi_i(t), \quad \dot{y}_i(t) = \sin\psi_i(t), \quad \forall i \in N \\ \dot{x}_P(t) = \cos\theta(t), \quad \dot{y}_P(t) = \sin\theta(t), \end{aligned} \tag{1}$$

Without loss of generality, the target is considered to be at origin, and this assumption can be relaxed. The game is considered to be terminated if (a) all the evaders are captured by the pursuer and (b) at least one evader reaches the target before being intercepted. In this work, the termination condition (a) is considered, and the optimal strategies are found. Within this framework, two extreme cases are possible: (i) the evaders cooperate and try to reach the target (CTGP), or they act independently with respect to each other and try to invade the target (NCTGP). For the CTGP, the cost function that captures the interaction of the game is given by:

$$J(\psi, \theta) = \sum_{i \in N} ||\mathbf{x}_{i_f}||, \tag{2}$$

where $\psi := (\psi_1, ..., \psi_n)$ and $\mathbf{x}_{i_f}$ is the capture state of the evaders away from the target. The evaders collectively try to minimize this cost, and the pursuer aims to maximize it. However, for the NCTGP, each evader E_i individually tries to minimize the cost function

given by:

$$J_i(\psi_i, \theta) = ||\mathbf{x}_{i_f}||. \tag{3}$$

and the pursuer tries to maximize the cost given in 2.

2.2 Game of Kind and Game of Degree for evaders

Game of Kind (GOK) refers to the partitioning of the combined state space into evader and pursuer-winning regions. All the initial positions in the state space are classified into the pursuer-winning region if there exists a strategy for the pursuer that guarantees a win independent of the evader team strategies. Similarly, all the initial positions in the state space are classified into the evader winning region if there exists a strategy for the evader team that guarantees a win independent of the pursuer team strategies. Further, the initial positions that result in a pursuer and an evader reaching the target simultaneously are also considered as a pursuer-winning region. For the 1v1 case, the GOK is given by the following proposal:

Lemma 2.1 *Consider the function* $B_i : \mathbb{R}^4 \to \mathbb{R}$ *given by:*

$$B_i(\mathbf{x}) = R_i - R_P, \tag{4}$$

where $R_i = ||\mathbf{x}_i||$ *and* $R_P = ||\mathbf{x}_P||$. *The pursuer wins if* $B_i(\mathbf{x}) \geq 0$ *and the evader wins if* $B(\mathbf{x}) < 0$.

This says that when the agents have uniform and same speed, the pursuer wins the game if it is closer to the target and vice versa. Using this lemma, the GOK for the 1vn case can be found and is given by the following proposal.

Theorem 2.2 *Consider the function* $B : \mathbb{R}^{2(n+1)\to\mathbb{R}}$ *given by:*

$$B(\mathbf{x}) = \min_{i \in N} B_i(\mathbf{x}_P, \mathbf{x}_i) \tag{5}$$

where B_i *is defined in equation* (4). *The pursuer wins if* $B(\mathbf{x}) \geq 0$ *and the evaders win if* $B(\mathbf{x}) < 0$.

Hence, the pursuer is guaranteed to win if the barrier function defined in equation (5) takes a nonnegative value for a particular initial condition.

Having found the pursuer and evader winning region and the focus being the pursuer winning region, the Game of Degree (GOD) refers to finding the optimal strategies for the agents within the interested region. First, the GOD for the evaders is considered.

Given that there are two scenarios CTGP and NCTGP, we would expect the evaders to have different strategies in the pursuer winning region. However, with the associated Hamiltonian for CTGP given as:

$$\begin{aligned} H &= \sum_{i \in N} (\lambda_{x_i} \dot{x}_i + \lambda_{y_i} \dot{y}_i) + \lambda_{x_P} \dot{x}_P + \lambda_{y_P} \dot{y}_P \\ &= \sum_{i \in N} (\lambda_{x_i} \cos\psi_i + \lambda_{y_i} \sin\psi_i) + \lambda_{x_P} \sin\theta + \lambda_{y_P} \sin\theta, \end{aligned} \tag{6}$$

where $\lambda := (\lambda_{x_P}, \lambda_{y_P}, \lambda_{x_n}, \lambda_{y_n}, ..., \lambda_{x_n}, \lambda_{y_n})$ denotes the costate vector and the associated Hamiltonian for the coupled optimal control problem in NCTGP given by:

$$\begin{aligned} H_i &= \lambda_{x_i} \dot{x}_i + \lambda_{y_i} \dot{y}_i = \lambda_{x_i} \cos\psi_i + \lambda_{y_i} \sin\psi_i, \\ H_P &= \lambda_{x_P} \dot{x}_P + \lambda_{y_P} \dot{y}_P = \lambda_{x_i} \cos\theta + \lambda_{y_i} \sin\theta, \end{aligned} \tag{7}$$

where $\lambda_i = (\lambda_{x_i}, \lambda_{y_i})$ is the costate associated with E_i and $\lambda_P = (\lambda_{x_P}, \lambda_{y_P})$ is the costate associated with P, it is seen that the hamiltonians are independent of the state and the costates remain constant. Further, the costates satisfy the conditions due to the structure of the terminal cost in CTGP and NCTGP. Through this, it can be shown that in the both scenarios, the optimal strategies for each Evader E_i is given by:

$$\cos\psi_i^\star = \frac{-\lambda_{x_i}}{\sqrt{\lambda_{x_i}^2 + \lambda_{y_i}^2}}, \quad \sin\psi_i^\star = \frac{-\lambda_{y_i}}{\sqrt{\lambda_{x_i}^2 + \lambda_{y_i}^2}}, \tag{8}$$

Hence, it is seen that irrespective of the scenario, the optimal strategies for each Evader E_i remain the same. Since the pursuer aims to maximise the cost function, which is of similar form in both scenarios, the optimal strategies for the pursuer also remain the same, irrespective of the scenario. However, the optimal strategy (θ^*) for the pursuer cannot be directly found as the corresponding costates take 0 value and thus θ^* can potentially take

any value. Further, since the costates remain constant in both CTGP and NCTGP, the optimal strategy remains constant, so under optimal play, all the agents travel in a fixed heading direction.

Since it doesn't depend on whether the evader behaves independently or not, the optimal strategy for them can be obtained from the 1v1 case in the pursuer-winning region. It can be shown that the Value function given by

$$\mathcal{V}(\mathbf{x}) = \frac{my_C + x_C}{\sqrt{1+m^2}}, \tag{9}$$

where $m = \frac{y_i - y_P}{x_i - x_P}$, $x_C = (x_i + x_P)/2$ and $y_C = (y_i + y_P)/2$ which talks about the distance of the bisector line of the pursuer and evader from the target is the payoff consequent of the play, and is a value function candidate that solves the HJI-PDE for this 1v1 game given by

$$\min_{\psi_i} \max_{\theta} \left[\frac{\partial \mathcal{V}}{\partial x_P} \cos\theta + \frac{\partial \mathcal{V}}{\partial y_P} \sin\theta + \frac{\partial \mathcal{V}}{\partial x_i} \cos\psi_i + \frac{\partial \mathcal{V}}{\partial y_i} \sin\psi_i \right] = 0. \tag{10}$$

This boils down to the optimal strategy for both the pursuer and the evader in this 1v1 situation to head to the point on the perpendicular bisector of the line segment joining the two agents that is closest to the target.

2.3 Optimal Strategy for Pursuer and Convexity analyses

Since each evader has no motivation to cooperate, they follow the strategy for the 1v1 scenario. However, the pursuer that aims to capture all the evaders away from the target has to solve a multi-objective optimisation problem given by

$$\max_{\theta} \begin{bmatrix} ||\mathbf{x}_{1_f}|| \\ \vdots \\ ||\mathbf{x}_{n_f}|| \end{bmatrix}. \tag{11}$$

Now considering $I_1, I_2, ..I_n$ as the interception points in the 1v1 game between P and E_i, and the heading directions denoted by $\psi_1^\star, .., \psi_n^\star$ and $\theta_1^\star, ..\theta_2^\star$ respectively for the

evaders and the pursuer, it should be noted that all the terms $I_i, \psi_i^\star, \theta_i^\star$ are instantaneous quantities depending on the state position at the current instant. Since following optimally with respect to one evader will make it sub-optimal for other evaders, there is a need to come up with an appropriate θ. To this end, considering the strategies of the evaders fixed, we define $d_i(\theta) := ||\mathbf{x}_{i_f}||\big|_{\psi_i=\psi_i^\star}$ and $d(\theta) := [d_1(\theta), ..., d_n(\theta)]^T$. Thus, the multi-objective optimization problem considered in (11) can be written as

$$\max_{\theta} d(\theta) = \max_{\theta} \begin{bmatrix} d_1(\theta) \\ \vdots \\ d_n(\theta) \end{bmatrix}. \tag{12}$$

Now, considering the notion of pareto-optimality, it can be shown that the pursuer heading at an angle in the set $[\min_{i\in N} \theta_i^\star, \max_{i\in N} \theta_i^\star]$ maps to a point in the pareto frontier with respect to the optimization problem considered in (12).

Any point in this multi-objective pareto frontier can be a solution to equation (12), but to choose one direction, a scalar objective function is to be defined for which solutions can be found easily. To this end, the function d_i on which the objective function is built is shown to be concave below.

Theorem 2.3 *The function $d_i : S^1 \to \mathbb{R}\ \forall i \in N$ is a concave function of the heading angle of the pursuer.*

At any given instance of the game, let the game geometry be depicted as in Figure 1. Without loss of generality, let the x-axis be defined on the line connecting P and E, and the y-axis be the perpendicular bisector of the line PE. Let $P's$ coordinates be $(k, 0)$, $E's$ be $(-k, 0)$ and $T's$ be (x_t, y_t), with $y_t > 0$. Let the pursuer move a distance r in time interval Δt, with a heading angle θ. The new position of the pursuer will be P_1 with co-ordinates $(k - rcos(\theta), rsin(\theta))$. Since, the pursuer and the evader have same speed, let the evader also moves a distance r in time interval Δt in the optimal heading direction $\psi_i^\star$. Note that $\psi_i^\star$ is measured anticlockwise from the positive x-axis and $\tan\psi_i^\star = \frac{y_t}{k}$. The new position of the evader is now E_1 with coordinates $(-k + r\cos\psi_i^\star, r\sin\psi_i^\star)$.

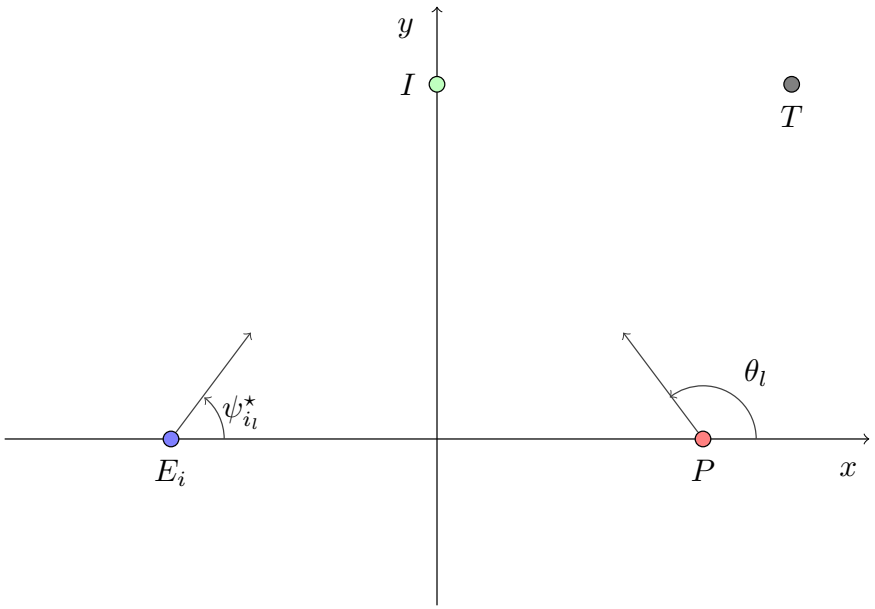


Figure 1: The geometry of the game with P and E_i

$$\text{Mid point of } P_1E_1 = \left(\frac{r(\cos\psi_i^\star + \cos\theta)}{2}, \frac{r(\sin\theta + \sin\psi_i^\star)}{2}\right) \tag{13}$$

$$\text{Slope of } P_1E_1 = \frac{r(\sin\theta - \sin\psi_i^\star)}{2k + r(\cos\theta - \cos\psi_i^\star)} \tag{14}$$

Slope of perpendicular bisector of P_1E_1

$$m = \frac{-(2k + r(\cos\theta - \cos\psi_i^\star))}{r(\sin\theta - \sin\psi_i^\star)} \tag{15}$$

The equation of the perpendicular bisector is now given by $y = mx + c$ where c can be obtained from the midpoint of the line P_1E_1. So the equation is

$$\begin{aligned} 2r(\sin\theta - \sin\psi_i^\star)y + (4k + 2r(\cos\theta - \cos\psi_i^\star))x \\ -2kr(\cos\theta + \cos\psi_i^\star) = 0. \end{aligned} \tag{16}$$

The distance from the target to the perpendicular bisector of P_1E_1 is now given by

$$\begin{aligned} d(\theta) = &[(4k + 2r(\cos\theta - \psi_i^\star)x_t + 2r(\sin\theta - \sin\psi_i^\star)y_t \\ &- 2kr(\cos\theta + \cos\psi_i^\star))]\Big/[16k^2 + 4r^2(\cos^2\theta^2 + \cos^2\psi_i^\star \\ &- 2\cos\theta\cos\psi_i^\star) + 16kr(\cos\theta - \cos\psi_i^\star) \\ &+ 4r^2(\sin^\theta + \sin^2\psi_i^\star - 2\sin\theta\sin\psi_i^\star)]^{\frac{1}{2}}. \end{aligned} \tag{17}$$

By taking $\Delta t \to 0$, we have $r \to 0$, then this function can be approximated as

$$\begin{aligned} d(\theta) &\approx \frac{4kx_t + 2r(\sin\theta - \sin\psi_i^\star)y_t - 2kr(\cos\theta + \cos\psi_i^\star)}{4k} \\ &= x_t + \frac{r}{2k}(\sin\theta - \sin\psi_i^\star)y_t - \frac{r}{2}(\cos\theta + \cos\psi_i^\star). \end{aligned} \tag{18}$$

A critical point for the function $d(\theta)$ is obtained where the derivative is zero.

$$\begin{aligned} \frac{d}{d\theta}d(\theta) = 0 &\implies y_t\cos\theta^\star + k\sin\theta^\star = 0 \\ &\implies \tan\theta^\star = -\frac{y_t}{k} \end{aligned} \tag{19}$$

This implies that if P heads towards the intercept point $I(0, y_t)$, the function d attains an extrema. To find whether it is a minima or a maxima, we can write

$$\begin{aligned} d(\theta) &= x_t + \frac{r}{2k}\left(y_t(\sin\theta - \sin\psi_i^\star) - k(\cos\theta + \cos\psi_i^\star)\right) \\ &= x_t + \frac{r\sqrt{y_t^2 + k^2}}{2k}\left(\sin\psi_i^\star(\sin\theta - \sin\psi_i^\star)\right. \\ &\quad \left. - \cos\psi_i^\star(\cos\theta + \cos\psi_i^\star)\right) \\ &= x_t + \frac{r\sqrt{y_t^2 + k^2}}{2k}\left(-\cos(\theta + \psi_i^\star) - 1\right). \end{aligned} \tag{20}$$

and this can be shown to reach a maxima in the domain of interest.

Note that this is in the local frame and in the global coordinates, the distance can be written as

$$d(\theta) = T_x + \frac{r\sqrt{T_y^2 + k^2}}{2k}\left(-\cos(\theta + \psi_i^\star - 2\delta) - 1\right). \tag{21}$$

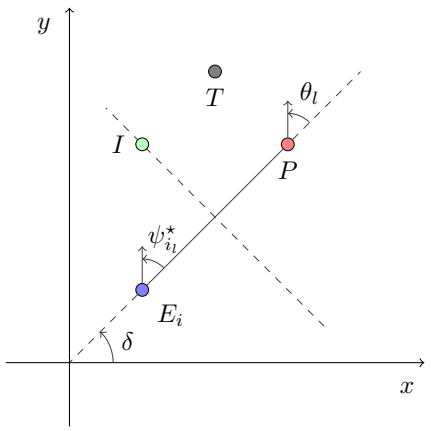


Figure 2: The geometry of the game with P and E_i

where T_x and T_y represent the distance of the target from the bisector line and the distance of the target from the Evader-Pursuer line, respectively. Further, k represents the distance of agents from the bisector line, and δ is the angle of Evader to Pursuer line from the x-axis, which is clearly shown in Figure2. So, d is concave for $\theta + \psi_i^\star - 2\delta \in [\pi/2, 3\pi/2] \implies \theta \in [\pi/2 - \psi_i^\star + 2\delta, \pi/2 - \psi_i^\star + 2\delta]$. Note that $\theta^\star = \pi - \psi_i^\star$ from (19), which implies that the pursuer must head directly towards the interception point. Thus, the current result is consistent with the results established before.

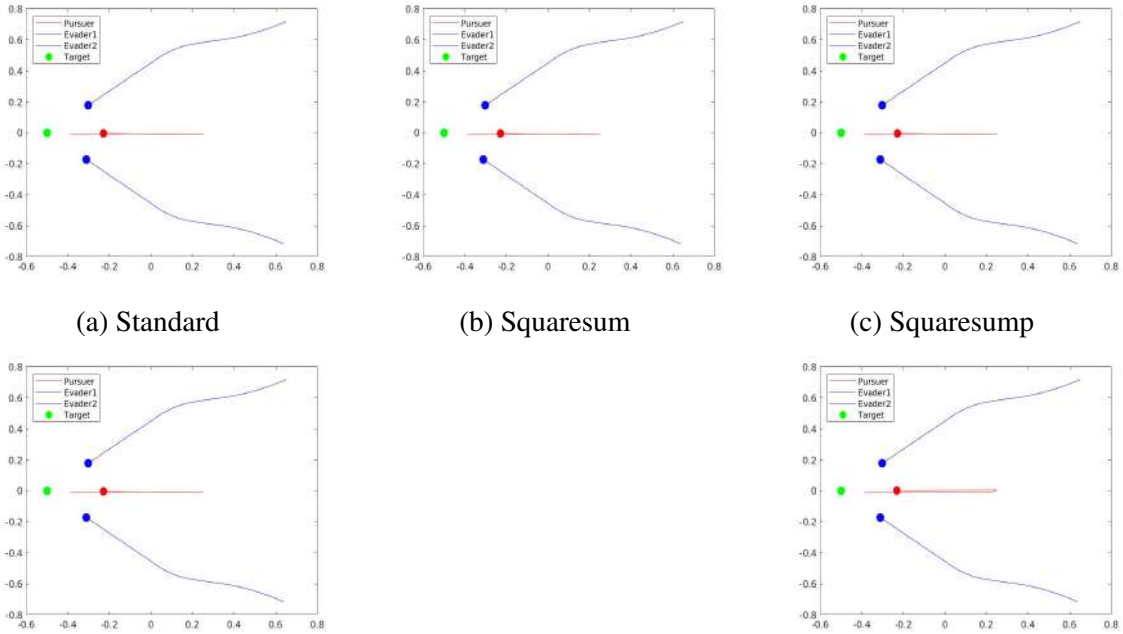
In summary, in this Theorem we show that every d_i is concave in its heading angle θ where $\theta \in [\theta_i^\star - \pi/2 + 2\delta, \theta_i^\star + \pi/2 + 2\delta]$. This then implies that the concavity of the function d_i holds in a set in $(-\pi, \pi]$ that extends $\pi/2$ degrees on either direction of the angle $\theta_i^\star + 2\delta$.

2.4 Analysis and study of Scalar Objective functions

The result in the previous section showed that d_i is concave around the optimal direction for the pursuer in a 1v1 game. However for a 1vn game, it can be the shown that d_i is concave in $\Theta^\star := \Theta^C \cap [\theta_1^\star, \theta_n^\star]$ where Θ^C is given by $[\theta_n^\star - \pi/2 + 2\delta, \theta_1^\star + \pi/2 + 2\delta]$. Hence, the search for a single angle can be restricted to $\Theta^\star$. The multi-objective optimisation problem considered in equation (12) needs to be converted into a scalar optimisation problem, and a unique solution needs to be found. In this regard, certain objective functions of d_i are considered, and their analysis for various initial conditions and computational efficiency is considered below.

1. **Squaresum** $\theta^\star = \underset{\theta \in \Theta^\star}{\arg\max} \sum_{i \in N} d_i^2(\theta)$
2. **Sum** $\theta^\star = \underset{\theta \in \Theta^\star}{\arg\max} \sum_{i \in N} d_i(\theta)$
3. **Standard** $\theta^\star = \underset{\theta \in \Theta^\star}{\arg\min} \sum_{i \in N} \frac{1}{d_i(\theta)}$
4. **Squaresump** $\theta^\star = \underset{\theta \in \Theta^\star}{\arg\min} \sum_{i \in N} \frac{1}{d_i^2(\theta)}$
5. **Heuristic** $\theta^\star = \underset{\theta \in [\min_{i \in N} \theta_i^\star, \max_{i \in N} \theta_i^\star]}{\arg\max} \sum_{i \in N} w_i(\theta) p_i(\theta)$ where $w_i(\theta) = \frac{p_i(\theta)}{\sum_{i \in N} p_i(\theta)}$,

Initial condition 1

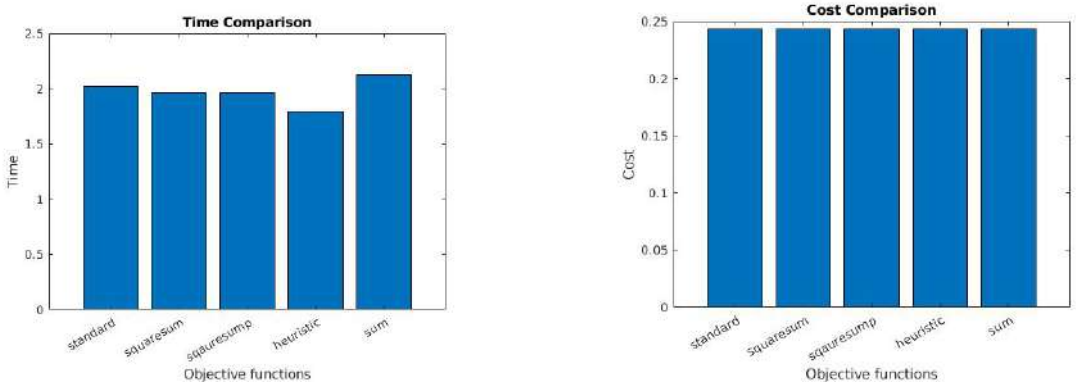


(a) Standard (b) Squaresum (c) Squaresump

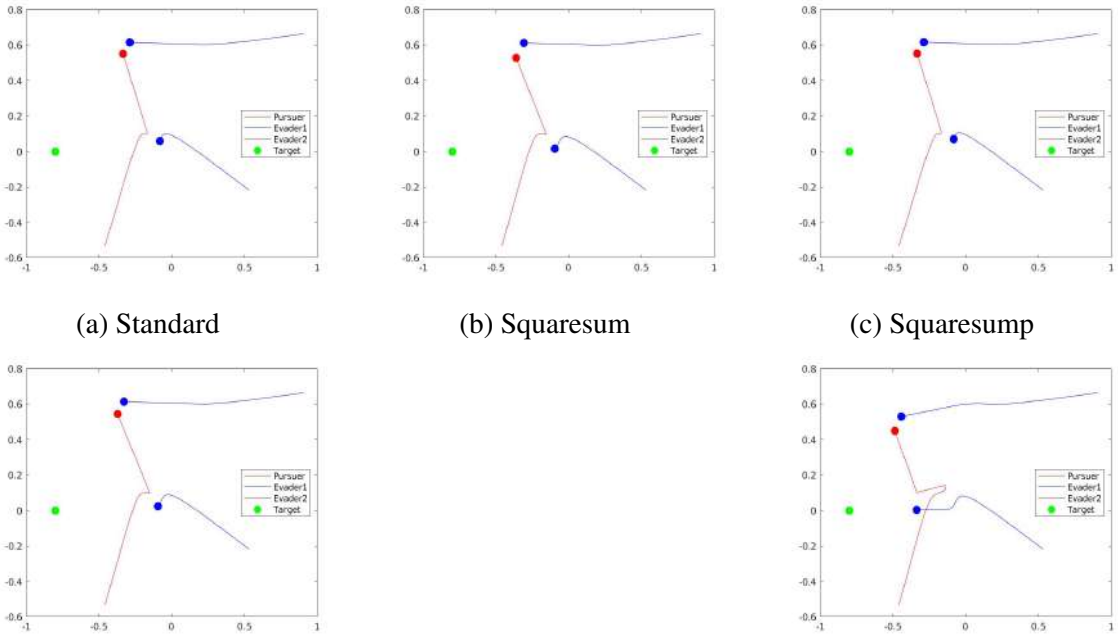
(d) Sum (e) Heuristic

Figure 3: Trjactories for a symmetrical initial condition

It can be seen that the trajectories of all objective functions look almost similar, and so for this initial condition, all objective functions have similar performance. This is quantitatively seen in the following cost comparison bar plot. However, the heuristic has lower time complexity, which is evident in the time comparison bar plot.



Initial condition 2

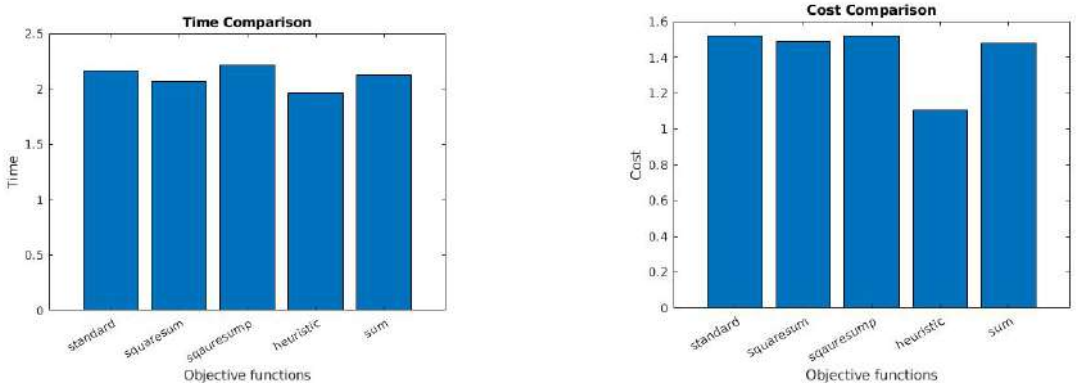


(a) Standard (b) Squaresum (c) Squaresump

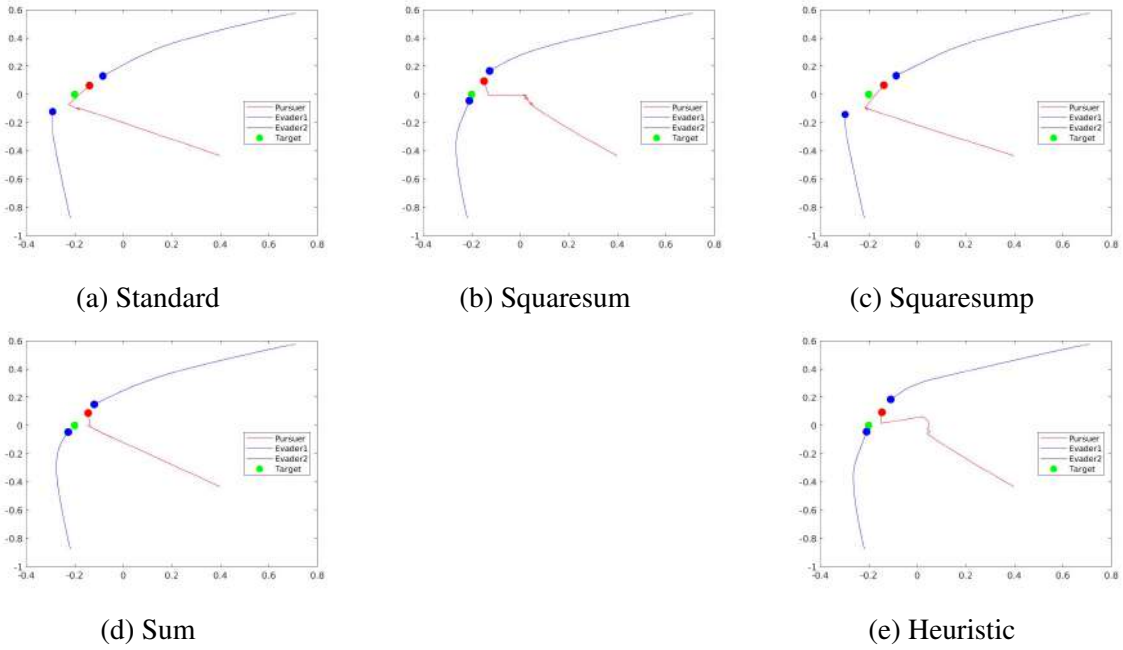
(d) Sum (e) Heuristic

Figure 5: Trjactories for Away capture from target

It can be seen that the trajectories of all objective functions look almost similar except for the heuristic-based feedback law, and so for this initial condition, all feedback law based on minimising or maximising a concave or convex objective functions respectively have similar performance while the heuristic performs poorly. This is quantitatively seen in the following cost comparison bar plot. However, the heuristic has lower time complexity, which is evident in the time comparison bar plot.



Initial condition 3

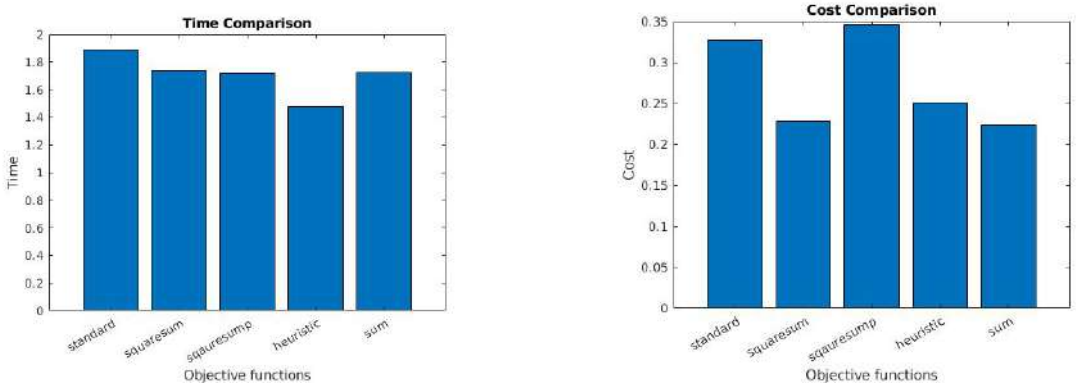


(a) Standard (b) Squaresum (c) Squaresump

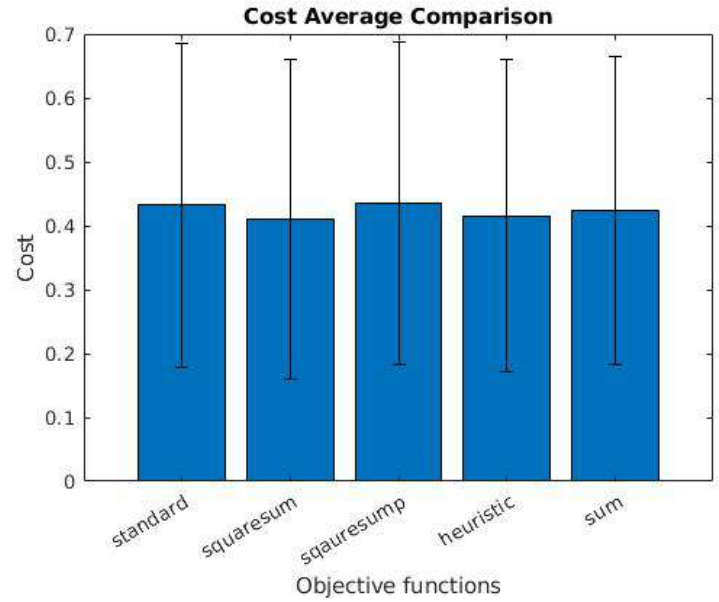
(d) Sum (e) Heuristic

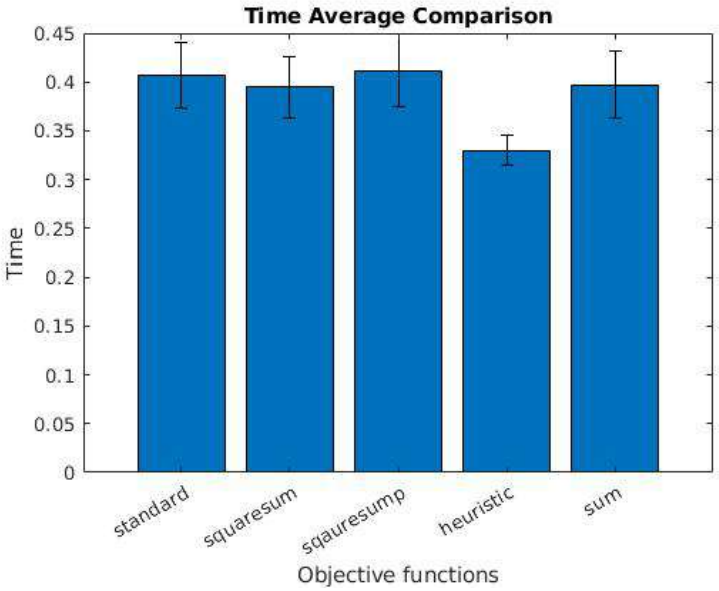
Figure 7: Trjactories for Away capture from target

In this case, the different objective functions have different performances, and the square-sump turns out to be the best, but the heuristic turns out to be the best in time comparison, as expected.



Since different initial conditions give different performances, 50 experiments with random initial conditions were considered and the averaged time and cost comparisons are as follows.





2.5 Hardware Implementation details

The above-mentioned simulations were experimented with hardware, and the results are shown. Three main components of this hardware implementation are detailed in the following subsections.

2.5.1 Omni-Direction EV3 Robot

The given agents in the game are point mass agents, which are omnidirectional. To this end, to replicate the behaviours of the simulated agents, it was necessary to utilise physical counterparts that mirrored their attributes. Hence, a decision was made to employ omnidirectional robots for real-world experiments, enabling direct comparisons to simulation outcomes. The Lego Mindstorms EV3 platform was chosen to build these omnidirectional robots due to its exceptional versatility, technical specifications, and adaptability.[1][2]
The Lego Mindstorms EV3 is a sophisticated robotics kit renowned for its modular design, enabling users to construct robots tailored to specific tasks and scenarios. Equipped with a powerful ARM9 processor and a comprehensive array of sensors, including touch, colour, and infrared, the EV3 offers a robust foundation for experimentation and development.

One of the standout features of the EV3 is its firmware, which provides a stable and efficient operating environment for executing complex algorithms and controlling various actuators. This firmware supports advanced functionalities such as Wi-Fi connectivity, allowing seamless integration with other devices and enabling remote operation and data transfer.
Moreover, the customizability offered by EV3 enabled easy tinkering in building the omnidirectional robots when there was a deficit of required parts. Further, the extensive support and compatibility with Software tools such as **MATLAB**, made the hardware experiments easier and all the computations were run on a standalone computer, with just the motor commands sent to the robot. MATLAB Support Package for LEGO MINDSTORMS EV3 Hardware add-on was used in Matlab and the following commands were used to control an individual motor.

Figure 10: Lego-Mindstorms EV3 platform

```
myev3  = legoev3('Wifi',<IP address>,<robotid>);
mtr = motor(myev3,<output port>);
start(motor);
motor.speed = <desired speed>;
stop(motor)
```

The platform was communicated to the system to execute the MATLAB code using Wi-Fi and the Netgear WNA1100 n150 wireless USB adapter for the same.

Figure 11: Netgear WNA1100 wireless adapter

2.5.2 Motion-Capture System

In robotics systems, observing the system's state is required, as the control law is in the form of state feedback. State observation in the global frame is paramount for understanding and analyzing the behaviour of robots in real-world scenarios. Motion capture systems serve as invaluable tools in this regard, enabling precise tracking and recording of the robot's movements with respect to a fixed reference frame. These systems typically employ markers strategically placed on the robot's body to facilitate accurate localization and pose estimation. The placement of markers is a crucial consideration, as it directly impacts the quality and reliability of the captured data. Optimal marker placement ensures maximum visibility and distinguishability, minimizing occlusions and ambiguities during motion tracking.

(a) Motive Flex 3

(b) Motion Capture System in the lab

Motive Flex3, available in the laboratory, was used in the experiments. It presents a sub-millitmeter accuracy, making it well-suited for capturing intricate movements and subtle nuances in robot behavior.The system operates on the principle of optical tracking, wherein specialized cameras detect the positions of reflective markers in three-dimensional space. By triangulating the reflections from multiple cameras, the system calculates the precise locations of the markers, thereby reconstructing the robot's motion in real-time. Motive Flex3 uses passive reflection for marker detection. Passive reflection involves the use of markers that are inherently reflective, typically coated with materials such as retro-reflective tape or paint. These markers passively reflect incident light back towards

the source, allowing specialized cameras within the motion capture system to detect and triangulate their positions with high precision. passive reflection facilitates non-intrusive tracking, as the markers do not require their own power source or active signaling components. This inherent simplicity makes passive reflection markers ideal for integration into robotic systems, where minimizing additional weight and complexity is paramount.

2.5.3 Robot Operating System

ROS (Robot Operating System) is a versatile framework widely adopted in robotics research and development, offering a comprehensive suite of tools and libraries for building complex robotic systems. At the heart of ROS lies its communication infrastructure, which revolves around the concept of ROS topics. ROS topics serve as the backbone for inter-process communication, enabling different nodes within a ROS system to exchange data seamlessly. Nodes can publish messages to topics, and other nodes can subscribe to these topics to receive the messages. This decoupled architecture fosters modularity and flexibility, allowing developers to design distributed systems where nodes can be added, removed, or replaced with ease. ROS topics facilitate the exchange of various types of data, including sensor measurements, control commands, and state information, facilitating collaboration and integration across different components of robotic systems..VRPN (Virtual Reality Peripheral Network) is a software framework commonly used for tracking objects in virtual reality and robotics applications. Within the ROS ecosystem, the $vrpn\_client\_ros$ package serves as a bridge between VRPN and ROS, allowing ROS nodes to subscribe to the pose and tracking data provided by VRPN servers.

The Motive system generates the rigid body's position and orientation. This is transmitted through the VRPN network. The $vrpn\_client\_ros$ package is used to receive the information broadcasted by the motive system and publish it in a ROS topic. This information of the robots' position and orientation is subscribed by the MATLAB node and this state feedback is used in the control feedback law, which is then converted as motor commands and transmitted to the robots via Wi-Fi. The following picture summarises the entire pipeline along with data flow.

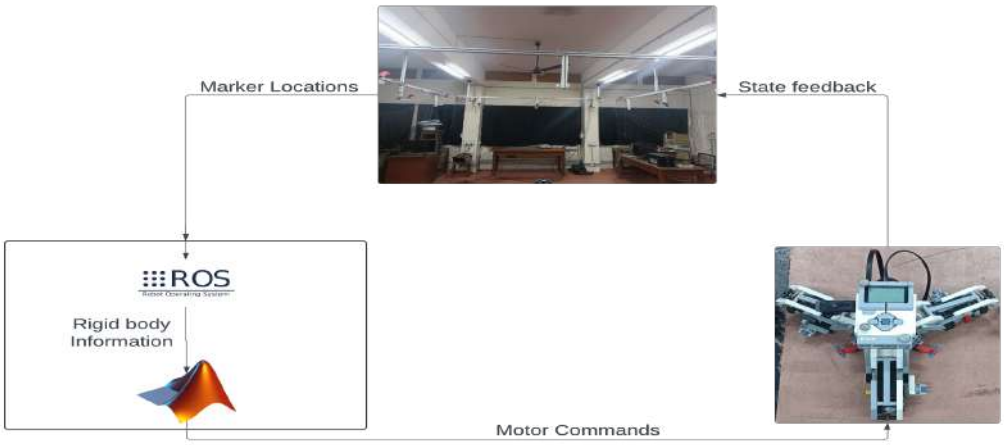


Figure 13: Data Flow diagram

2.5.4 Model of Omni-Directional Robot

The control of the real robots required the conversion of the control feedback law which was available as velocity in the form

$$\begin{aligned} \dot{x}_i(t) = \cos\psi_i^\star(t), \quad \dot{y}_i(t) = \sin\psi_i^\star(t), \quad \forall i \in N \\ \dot{x}_P(t) = \cos\theta^\star(t), \quad \dot{y}_P(t) = \sin\theta^\star(t), \end{aligned} \tag{22}$$

where (x_P, y_P) and (x_i, y_i) refers to the position of the pursuer P and evader E_i respectively in the global frame. To this end, the standard omni-directional model was used and the Frame transformations were used to convert the global velocity to local velocity and then later this was converted to motor commands, the details of which are as follows.
For velocity given by (v_x^g, v_y^g) in the global frame, the relation to (v_x^l, v_y^l), velocity in the local frame in 2D space is given by:

$$\begin{bmatrix} v_x^l \\ v_y^l \end{bmatrix} = \begin{bmatrix} cos\phi & sin\phi \\ -sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} v_x^g \\ v_y^g \end{bmatrix} \tag{23}$$

where ϕ refers to the orientation of the robot in the global frame. Further the relation to convert the robot's local frame velocity to the motor's angular velocity in a three wheeled

omni-directional robot is given by:

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} -d & 1 & 0 \\ -d & -1/2 & -sin(\pi/3) \\ -d & -1/2 & -sin(\pi/3) \end{bmatrix} \begin{bmatrix} \omega_z^l \\ v_x^l \\ v_y^l \end{bmatrix} \tag{24}$$

where r refers to the wheel radius and d refers to the wheel to centre distance which are 0.025m and 0.15m respectively in the Lego EV3 robot. ω_z^l refers to the angular velocity of the robot. Since it is an omni-directional robot, ω_z is taken to be 0. These angular velocities are sent to the robot using the MATLAB command:

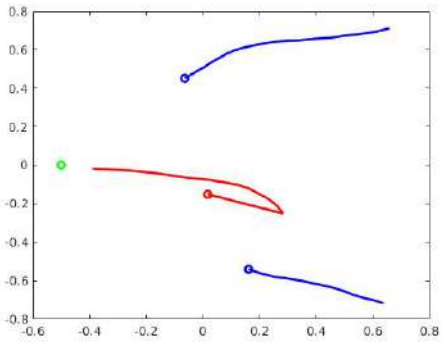
```
motor.speed = min(max(u,-80),80);
```

Here, u refers to the angular velocity of a single wheel. Since the maximum allowed range for the Lego Ev3 robot was [-100 to 100]. The velocities were clipped at -80 to 80 to reduce power consumption. This was not a problem in most cases as it was noticed that the angular velocities didn't cross the -80 to 80 range. However, even in cases where it crossed, minimal errors popped up, but because of the closed loop system, the behaviour of the agents and the outcome of the game did not change.

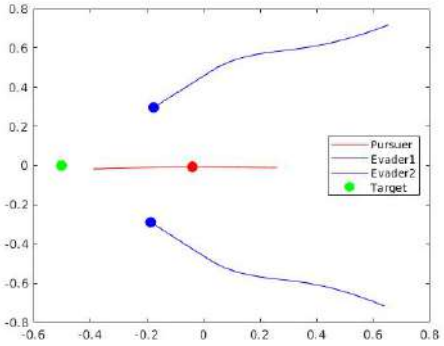
2.6 Hardware Results

The hardware experiments were done for a 1v2 game and its results are shown for two different initial condition along with the corresponding plots in simulation.

Initial Condition : Symmetrical Evaders

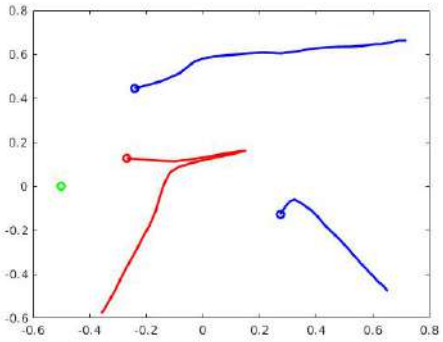


(a) Hardware result

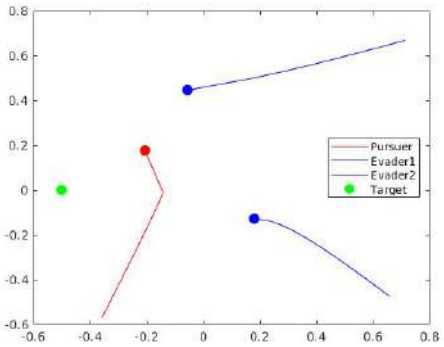


(b) Simulation result

Initial Condition : Aymmetrical Evaders



(a) Hardware result



(b) Simulation result

It should be noted that the value of r and the termination condition were kept constant for both the simulation and hardware result, and it can be seen from the plots that although the plots are not exactly matching, the agents follow similar strategies in both simulation and hardware and we can see that the game conditions are almost similar.

2.6.1 Miscellaneous Problems and their solutions

The entire pipeline given its closed-loop form helped in keeping the constancy of the result of the game. However, in the real world scenario many irregularities and drifts were caused. Some of the key problems are mentioned below:

- **Rigid-body mismatch**: Since the agents were homogeneous, similar marker placements on the agents resulted in Motive software confusing between the rigid bodies and sending position feedback in the wrong order. This was solved using dissimilar marker placements on the agents.
- **Orientation feedback mismatch**: Initially the markers were placed in symmetrical fashion on each agents. However, this led to the motive software confusing a single corner of the agent and this resulted in the orientation feedback being wrong and added with a value of π at times. This was solved with irregular and unsymmetrical marker placements on each agent and the following picture shows these agents.

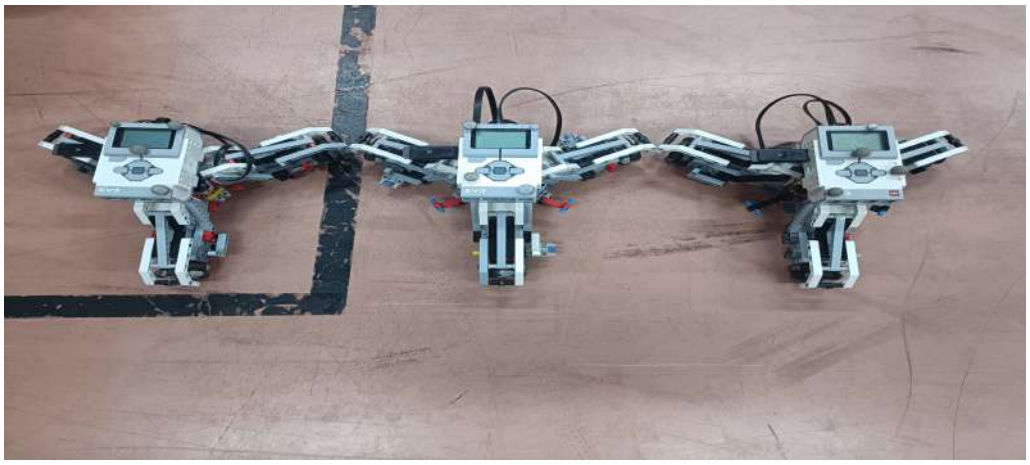


Figure 16: Agents with asymmetrical and dissimilar marker arrangements

- **Poor Odometry**: In order to improve upon the estimation of the position feedback of the agents, it was proposed to use odometry to improve the prediction made by the motion capture system, however due to the model mismatch caused due to irregularities in the real world (slip on wheels, slight mismatch in the sizes on wheels), the correction was worse. Hence, it was decided to just use motion capture and get the feedback at a higher frequency and conduct experiments during the night time.

3 Multi-Agent Reinforcement Learning for TGP

This chapter discusses the Multi-agent Reinforcement Learning formulation of Single-Pursuer, Single-Evader game, followed by the details of the simulation environment and ablation studies of parameters in the training and future directions.

3.1 General Formulation of MARL problems

Multi-Agent Reinforcement learning, a class of RL algorithms can be modeled as sequential decision making process known as discounted Markov decision process (MDP) whose formulation is given as follows:
A Multi-agent Markov Decision Process (MDP) is a mathematical framework used to model decision-making in environments where multiple agents interact with each other and their surroundings. It is defined by a tuple $(\mathcal{S}, \mathcal{A}_1, \ldots, \mathcal{A}_N, \mathcal{P}, \mathcal{R}_1, \ldots, \mathcal{R}_N, \gamma)$, where:

- $\mathcal{S}$ is the set of states. Each state $s \in \mathcal{S}$ represents a configuration of the environment.
- $\mathcal{A}_i$ is the set of actions available to agent i. Each agent i can take actions from its respective action space $\mathcal{A}_i$.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \times \mathcal{S} \to [0, 1]$ is the state transition probability function. It defines the probability of transitioning from state s to state s' given the joint action of all agents. Formally, $\mathcal{P}(s'|s, a_1, \ldots, a_N)$ represents the probability of transitioning to state s' when agents take actions $a_1, \ldots, a_N$ in state s.
- $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_1 \times \ldots \times \mathcal{A}_N \to \mathbb{R}$ is the reward function for agent i. It specifies the immediate reward received by agent i upon taking actions $a_1, \ldots, a_N$ in state s. The goal of each agent is typically to maximize its cumulative reward over time.
- $\gamma \in [0, 1]$ is the discount factor. It represents the preference for present rewards over future rewards. A discount factor of $\gamma = 0$ means only immediate rewards are considered, while $\gamma = 1$ means future rewards are considered equally to immediate rewards.

In Multi-agent Markov Decision Processes (MDPs), the primary objective is to find policies for each agent that maximize their cumulative rewards over time. A policy π_i for agent i is essentially a mapping from states to actions, i.e., $\pi_i : \mathcal{S} \rightarrow \mathcal{A}_i$. It represents the strategy or decision-making mechanism employed by each agent to interact with its environment.

The goal is to determine optimal policies π_i^* or value functions V_i^* that guide each agent's decision-making process towards achieving desirable outcomes. These optimal policies or value functions maximize the expected cumulative reward for each agent in the environment, thus leading to effective and efficient decision-making in multi-agent settings.

The value function $V_i^{\pi}(s)$ represents the expected cumulative reward that agent i can achieve starting from state s and following policy π_i. It is defined recursively as:

$$V_i^{\pi}(s) = \mathbb{E}_{\pi_i}\left[\sum_{t=0}^{\infty} \gamma^t \cdot \mathcal{R}_i(s_t, a_{i,t}) \mid s_0 = s\right] \tag{25}$$

Here, s_t is the state at time t, $a_{i,t}$ is the action taken by agent i at time t, and $\mathbb{E}_{\pi_i}$ denotes the expectation with respect to the policy π_i.

A variety of On-policy, off-policy, value-based and policy based algorithms have been used in Reinforcement Learning and their extensions have been found in MARL as well. However, in this work, given the competitive nature of the game and consideration of computational resources, Multi-Agent version of PPO (MAPPO)[15] has been used and the details of which are discussed in the next section.

3.2 On-Policy MAPPO

Policy gradient algorithms are a type of reinforcement learning algorithm that directly optimizes over the space of policy functions by maximizing the expected reward. These algorithms work by iteratively updating the policy based on the gradient of the expected reward with respect to the policy parameters. PPO is a popular on-policy policy gradient algorithm that addresses some of the limitations of standard policy gradient methods. PPO uses clipped surrogate objectives to ensure stable policy updates and prevent large policy

changes that can lead to performance degradation. Important factors of PPO:

- **Clipped Surrogate Objectives:** PPO introduces clipped surrogate objectives that restrict the policy update to stay close to the original policy. This clipping helps to prevent the policy from diverging significantly during training, which can lead to instability.
- **Importance Sampling:** PPO utilizes importance sampling to correct for the off-policy nature of updates in on-policy algorithms. This helps to reduce the variance of the policy gradient estimate.

MAPPO extends PPO to address the challenges of multi-agent reinforcement learning (MARL). In MARL, multiple agents interact with each other and the environment, making it difficult to directly optimize individual agent policies due to the non-stationary nature of the environment experienced by each agent. Training individual policies in a multi-agent setting can lead to suboptimal behavior due to the non-stationary nature of the environment. When one agent changes its policy, it can significantly impact the environment for other agents, requiring them to adapt their policies as well. This creates a feedback loop that can hinder convergence. MAPPO addresses this challenge by introducing a centralized critic and a policy actor for each agent. The centralized critic estimates the value function for the entire global state, while each agent's policy actor takes the local observation of that agent as input and outputs an action.However, it should be noted that the non-stationarity is an important problem who effects are shown in the results and a proposal is made to handle this issue, which is discussed in the future scope section. In summary MAPPO is a model-free, stochastic on-policy policy gradient CTDE (centralized training, decentralized execution) multi-agent algorithm that uses a centralized value function to estimate a single value that is used to guide the policy updates of all agents and the steps followed by the MAPPO algorithm are:

- **Policy Actors:** Each agent interacts with the environment and collects experience tuples containing state, action, reward, and next state information.
- **Centralized Critic:** The collected experience is used to train the centralized critic to estimate the state-value function.

- **Policy Update:** The policy actors are updated using the PPO clipped surrogate objective function, but instead of using the individual agent's advantage estimate, MAPPO uses the advantage estimate based on the centralized critic's value function.

The advantages are found using Generalised Advantage Estimation(GAE) which is a is a technique used in policy gradient algorithms to estimate the advantage function for each state-action pair. The advantage function measures how much better an action is compared to the average action taken by the policy in a particular state. Traditionally, the advantage function is estimated using the discounted sum of rewards following a state-action pair:

$$A_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i} - V(s_t) \tag{26}$$

The GAE advantage estimate is calculated as:

$$A_t = \sum_{i=0}^{k-1} \gamma^i \lambda r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t) \tag{27}$$

where:

* k is the truncation parameter (typically a small integer)

This equation incorporates the discounted sum of rewards for a truncated horizon k and then adds the discounted estimated value of the state at the end of the horizon. The λ parameter controls how much weight is given to past rewards compared to the estimated value function.

Further, MAPPO inherits its on-policy nature from Proximal Policy Optimization (PPO). On-policy algorithms learn from data collected by the current policy being evaluated.This characteristic offers MAPPO several advantages:

- **Stability:** On-policy methods tend to be more stable than off-policy methods, especially in environments with non-stationary or noisy data. Since on-policy methods update their policy based on the data they collect while following the current policy, they are less susceptible to issues such as divergence or instability.

- **Reduced Memory Footprint:** On-policy algorithms typically require less memory compared to off-policy methods.

3.3 Simulation Environment

Training reinforcement learning (RL) algorithms in simulation environments is crucial for several reasons. Simulations provide a controlled and safe environment for exploring and experimenting with RL algorithms without the risk of damaging physical systems or incurring costly mistakes. Additionally, simulations enable rapid prototyping and provide cheap data, allowing researchers and practitioners to test hypotheses and develop algorithms more efficiently. In this section, the simulation environment utilized for the experiments is described. The framework chosen for developing and running multi-agent simulations was PettingZoo . PettingZoo offers a diverse collection of environments suitable for studying various aspects of multi-agent systems.PettingZoo, an open-source Python library, provides a wide range of environments for multi-agent reinforcement learning research. It offers a unified interface for interacting with different environments, facilitating the development and testing of algorithms across various domains.The environments available in PettingZoo cover a broad spectrum of scenarios, including competitive, cooperative, mixed cooperative-competitive, and partially observable settings. Additionally, PettingZoo supports environments with discrete and continuous action spaces, allowing for versatile experimentation.

One of the environments available in PettingZoo, utilized extensively in the experiments, is the Multi-Agent Particle Environment (MPE). MPE is a simple multi-agent environment designed for benchmarking multi-agent reinforcement learning algorithms.In MPE, agents navigate a grid-world environment, interacting with each other and the environment to achieve specific goals. It supports scenarios involving cooperation, competition, and coordination among multiple agents, making it suitable for studying multi-agent coordination and communication strategies.The simplicity and versatility of MPE make it an ideal choice for prototyping and testing multi-agent algorithms. Its well-defined rules and clear objectives allow for straightforward experimentation and analysis.

Figure 17: Multi-Agent Particle Environment

However, in this work, a custom made MPE environment is made for a Single Pursuer-Single Evader TGP.

Figure 18: Multi-Agent Particle Environment

Here the agents are of the size 0.05 units of the pettingzoo environment and without any loss of generality, the target which is green in color is fixed at origin and the pursuer and evaders which are also of the same size are red and blue in color respectively and movable. It should be noted that in the work, the two agents have a maximum speed of 1 units, however they can accelerate and decelerate based on its need. The state of the agents is two dimensional and is of the form state of the form $\mathbf{x} := (x, y)$ representing its position in the two dimensional space. The environment follows the Gym environment

setupcite, which defines a standardized interface for interacting with reinforcement learning environments. This setup allows for easy integration with existing RL libraries and frameworks. The following functions are mandatory in this framework.

- **reset()**: Resets the environment to its initial state and returns the initial observations.
- **step(action)**: Takes an action as input and returns the next observations, rewards, and whether the episode has terminated.
- **render()**: Renders the current state of the environment for visualization purposes.
- **close()**: Cleans up resources associated with the environment, such as closing GUI windows.

These functions allow agents to interact with the environment, receive feedback in the form of observations and rewards, and perform actions based on their policies.

The observation and action spaces are both continuous and are structured as Box spaces within the Gym framework. Their shapes are determined by the necessary dimensions of the environment. Each element within these spaces is confined to a range between -1 and 1. The agents cannot push and pull each other. However, they can collide and based on certain collision termination conditions are defined which are discussed in the next section.

3.4 Ablation studies of important parameters

In this study, the state of each of the agent, was considered to be the position and the action for each of the policies are the velocities of the agents. It should be noted that the even the action space allows the velocity of a given agent to be 1.414 m/s, the maximum speed was capped at 1m/s and since the training is done in a closed-loop feedback, the training can handle this irregularity. There were certain important parameters which decided the performance of the policy learnt. It should be noted that even though the results shown are for sub-optimal policies, ablations were done to choose the best set of parameters. The set of parameter and their variations include:

- **Observations:** There were three different choices in the set of observations for each agent, which includes(shown for pursuer):
  - Local state observation only: $\mathbf{x} = (x_p, y_p)$
  - Local state appended with state of the other agent: $\mathbf{x} = ((x_p, y_p), (x_e, y_e))$
  - Local state appended with the relative state of the other agent: $\mathbf{x} = ((x_p, y_p), (x_e - x_p, y_e - y_p))$

  It was found that only the local observation confused the policy as for a given state of the pursuer, the evader state is not unique.

- **Memory Size:** Since it is an on-policy algorithm, the memory sizes were small. However, with the ablation studies it was found that, an optimal memory size was around the range of 40-60, as smaller memory sizes led to a reduced number of good transitions stored to find the advantages which were further used in updating the network weights and bigger memory sizes led to slowing down the training and led to poor convergence.

- **Reward functions:** Various combinations of reward terms were used for the agents and prominent ones include:
  - **Exponential position error:** $ke^{\|\mathbf{x}_a - \mathbf{x}_b\|}$ where $\mathbf{x}_a$ and $\mathbf{x}_b$ refers to the position of pursuer and evader respectively, in case of pursuer reward terms and evader and target respectively, in case of evader reward term.
  - **terminal rewards:** These terminal rewards were choosen based on making the game zero-sum and they were in effect during the termination conditions which are pursuer capturing evader and evader reaching target.
  - **Delay penalty:** A small negative delay penalty term of added to motivate the agent to complete the game in quick manner.

  It was found that when weight sums of these reward terms were added, the game often fell into a sub-optima and some desirable results in the subspace were found when the exponential term was not added, making the reward terms sparse.

4 Future Scope and Proposals

It can be seen from the previous section that Deep Reinforcement Learning inherently tries to solve a non convex optimisation problem, due to which the susceptibility in which it can fall into a local optima is high leading to undesirable behaviours. It is also seen that tedious reward shaping needs to be done, with lot of human effort spent in tuning the gains with different reward terms. This problems scales exponentially with more complex environments. Further, as it can be clearly seen in the reward curves, the Non-stationary problem is a crucial problem which needs to be solved to bring in required behaviours. To this end two different directions are proposed which will be pursued in the future as it was out of scope for this project duration.

4.1 Transfer learning

Given a source domain $\mathcal{M}_s$ and a target domain $\mathcal{M}_t$, transfer learning deals with learning an optimal policy for the target domain, by leveraging some information in the source domain. This can be in the form of the following:

- **Continual Learning:** Sequentially Learning multiple tasks without forgetting previously acquired knowledge.
- **Demonstrated Trajectories:** The target agent can learn from the behaviour of an expert.
- **Policy Distillation:** A set of teacher policies are learned on a set of source domains. A student policy is learned on a target domain by leveraging source policies.

These strategies can be used in MARL, where the trajectories generated by the analytical solutions of simpler two-dimensional agents can be used as expect demonstrations which the policies can learn to track. However, in complex environments which robots of different models, finding analytical solutions can be difficult, so the trajectories generated by these simpler models can provide an ansatz to optimal solution and strategies followed in cite ogmp can be used to finding optimal policies for each agents. This can also reduce the MARL problem into single agent problems and thereby preventing the non-stationarity

problem. Further, the entire state space can be divided in sub tasks and learnt individually, then continual learning can be used to learn a multi-modal policy which solves these individual sub-tasks based on the states of the system.

4.2 Alternating Policy Learning

Since Multiple agents learning the policy together changes the environment in single agents perspective, an interesting direction would be to alternative learn the policies for each agent. To give more clarification, starting with arbitrary policies for each agent, policy of one agent can be learnt for few episodes while the policy of the other agents put in pause. This can be switched for all the agents and they can be made to learn the game sequentially. This proposal was inspired from the idea of Generalised Policy Iteration, where the steps of policy evaluation and policy improvement happens alternatively and slow improvements lead to the optimal policies.

5 Conclusion

This thesis report has provided valuable insights into the implementation of model-based solutions for 1vN games in hardware and the analysis of different objective functions in finding optimal solutions for the pursuer. The exploration of various metrics has highlighted the importance of carefully selecting optimization criteria to achieve desired outcomes in multi-agent environments. Additionally, preliminary results of Multi-Agent Reinforcement Learning (MARL) on 1v1 games with relaxed assumptions have shown promise in tackling complex scenarios. However, these results remain subject to limitations, including simplified dynamics and homogeneous agent capabilities, and the non-stationary problem, necessitating further validation and extension in more realistic settings. Moving forward, the integration of model-based and model-free approaches offers a promising way to make systems better, blending planning and learning to create effective multi-agent systems.

References

[1] Lego-mindstorms ev3. URL https://www.lego.com/en-us/product/lego-mindstorms-ev3-31313.

[2] Omni-directional robot. URL https://makezine.com/projects/lego-holonomic-robot/.

[3] Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. URL https://www.marl-book.com.

[4] R. Bellman, R.E. Bellman, and Rand Corporation. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[5] P. Bernhard. Linear-quadratic, two-person, zero-sum differential games: Necessary and sufficient conditions. *Journal of Optimization Theory and Applications*, 27(1): 51–69, Jan 1979.

[6] J. V. Breakwell and P. Hagedorn. Point capture of two evaders in succession. *Journal of Optimization Theory and Applications*, 27(1):89–97, Jan 1979.

[7] Mo Chen, Zhengyuan Zhou, and Claire J Tomlin. Multiplayer reach-avoid games via pairwise outcomes. *IEEE Transactions on Automatic Control*, 62(3):1451–1457, 2016.

[8] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. AAAI Press, 2018. ISBN 978-1-57735-800-8.

[9] P. Hagedorn and J. V. Breakwell. A differential game with two pursuers and one evader. *Journal of Optimization Theory and Applications*, 18(1):15–29, Jan 1976.

[10] R Isaacs. Differential games: A mathematical theory with applications to warfare and pursuit, control and optimization. 1965.

[11] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[12] Kostas Margellos and John Lygeros. Hamilton-jacobi formulation for reach-avoid

differential games. *IEEE Transactions on Automatic Control*, 56, 11 2009. doi: 10.1109/TAC.2011.2105730.

[13] Meir Pachter and Yaakov Yavin. One pursuer and two evaders on the line: A stochastic pursuit-evasion differential game. *Journal of Optimization Theory and Applications*, 39:513–539, 1983.

[14] A. G. Pashkov and S. D. Terekhov. A differential game of approach with two pursuers and one evader. *Journal of Optimization Theory and Applications*, 55(2):303–311, Nov 1987. doi: 10.1007/BF00939087.

[15] C Yu, A Velu, E Vinitsky, Y Wang, A Bayen, and Y Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. arxiv 2021. *arXiv preprint arXiv:2103.01955*.