Online Prediction for Vision-Based Active Pursuit Using a Domain Agnostic Offline Motion Model

by

Sumedh Godbole

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2021 by the
Graduate Supervisory Committee:

Yezhou Yang, Chair
Siddharth Srivastava
Wenlong Zhang

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

In a pursuit-evasion setup where one group of agents tracks down another adversarial group, vision-based algorithms have been known to make use of techniques such as Linear Dynamic Estimation to determine the probable future location of an evader in a given environment. This helps a pursuer attain an edge over the evader that has conventionally benefited from the uncertainty of the pursuit. The pursuer can utilize this knowledge to enable a faster capture of the evader, as opposed to a pursuer that only knows the evader's current location. Inspired by the function of dorsal anterior cingulate cortex (dACC) neurons in natural predators, the use of a predictive model that is built using an encoder-decoder Long Short-Term Memory (LSTM) Network and can produce a more accurate estimate of the evader's future location is proposed. This enables an even quicker capture of a target when compared to previously used filtering-based methods. The effectiveness of the approach is evaluated by setting up these agents in an environment based in the Modular Open Robots Simulation Engine (MORSE). Cross-domain adaptability of the method, without the explicit need to retrain the prediction model is demonstrated by evaluating it in another domain.

ACKNOWLEDGEMENTS

For someone so excited about discovering new things and wanting to constantly work on interesting ideas, I consider myself incredibly lucky to have been afforded a chance at gaining exposure to research at such an advanced level. To be able to present a summary of what I have learned so far in the form of this thesis, is an absolute honor. There are people without whose help and guidance, this task would never have been possible. Their support, both academic and moral is what I strongly believe to have got me through tough times riddled with unprecedented difficulties.

I would like to begin by expressing sincerest gratitude to my advisor Dr.Yezhou Yang, without whose expertise, I would not have been able to get to where I am today as a researcher. His continuous support, invaluable advice, a commitment to always putting forth one's best efforts and a contagious zeal towards research has been deeply inspiring.

I would like to thank the members on my thesis committee, Dr.Sidhharth Srivastava and Dr.Wenlong Zhang for graciously agreeing to lend me their expertise toward bettering my work and taking it to the next level.

I would certainly be remiss if I do not acknowledge my friend and colleague Varun Jammula who is a doctoral candidate himself, for being there every step of the way. I will always be grateful for his guidance, help and the high amount of patience he has had to exhibit from time to time, working with me as a lab partner.

I am forever indebted to my family and friends for being my pillars of strength. I also thank anyone who has been kind enough to lend me their support in any capacity towards the completion of this thesis.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

To get us closer to the motivation behind this project, let us imagine the following scenario. The year is 3090. The world is now the epitome of technology where things we only aspire to be able to build today, are a thing of reality. Artificial Intelligence has taken over all walks of life. Autonomous Vehicles have occupied most of the automobile industry. Channel 9 News, which is now run solely by AIs is broadcasting an emergency. A human suspect has commandeered a vehicle for themselves and is fleeing the police. An autonomous police vehicle is close behind in chase (see Fig. 1.1). There are many factors making this job really challenging for the autonomous vehicle in pursuit. Firstly, the terrain this heated chase takes place on, has the potential to change rapidly as the suspect speeds away desperately, with the sole intent of getting away from the police as quickly as possible. Thus the vehicle pursuing it, has no way of knowing where the chase will lead them next. This renders the task of mapping those areas in advance very impractical to say the least. Secondly, the evader's movement is so reckless and random, they do not exhibit any recognizable patterns or stereotypical behavior. This means we cannot take the approach where said movement can be pre-learned as a behavioral trait. Moreover, we cannot ignore the fact that this scenario is one that has really high stakes involving several risks. A lot of entities besides just the pursuer and the evader, are indirectly involved and possibly at risk during the course of such a chase and the law enforcement vehicle in pursuit has to keep track of all such possibilities. The active traffic on the road is the first entity to be at risk from the ongoing chase. Collisions with these vehicles need to be avoided at all costs to prevent damages to property and more importantly loss of

civilian life. Moreover, even a small hindrance such as the pursuer having to slow down at the wrong time because of a minor collision, can result in the suspect ultimately shaking off the pursuit and getting away. This also means the AI pursuer failed at its primary task. This rules out many state-of-the-art reinforcement learning (RL) algorithms one might consider as a solution to be used with this futuristic pursuer involved in the chase, since they often have a significant percentage of episodes that simply end up in collisions. They also present several other drawbacks such as the need for pre-mapped environments and the sheer amount of processing power and time needed to develop such a solution.

Thus the preliminary question we seek the answer to is if the autonomous police pursuer that is chasing the randomly moving human suspect in the scenario described above, is able to overcome this seemingly insurmountable task and actually apprehend the suspect without any loss of life or any significant losses to property, what would constitute such an autonomous agent? What arsenal of reliable technology does it need to be supplied with, in order to give it the necessary edge to be successful in the chase? What would the overall system model of such a pursuer look like? If we aim to build such a pursuer, we surely take steps in the right direction leading to novel methodologies that can act as solutions to robust autonomous driving, even in the present day world. Therefore in an attempt to answer these questions we set up a project in the form of a Pursuit Evasion scenario. With obstacles mimicking the vehicles and other static entities involved in the chase, we simulate a previously unmapped environment consisting of two agents, an evader that moves erratically and a pursuer whose aim is to capture it.

To be able to implement an autonomous pursuit system that achieves consistent results in difficult scenarios like the one described above, is an enormously challenging task. For the strategy to be effective, the pursuing agent needs to be robust. It needs

the ability to function with limited knowledge of the domain, minimal equipment and a high degree of adaptability. Therefore in designing a pursuit system, we need to consider the effectiveness of the strategy being implemented. This effectiveness depends on several factors including but not limited to the evader, the pursuing agent, the environment and other static and dynamic actors that play a passive role in said scenario.

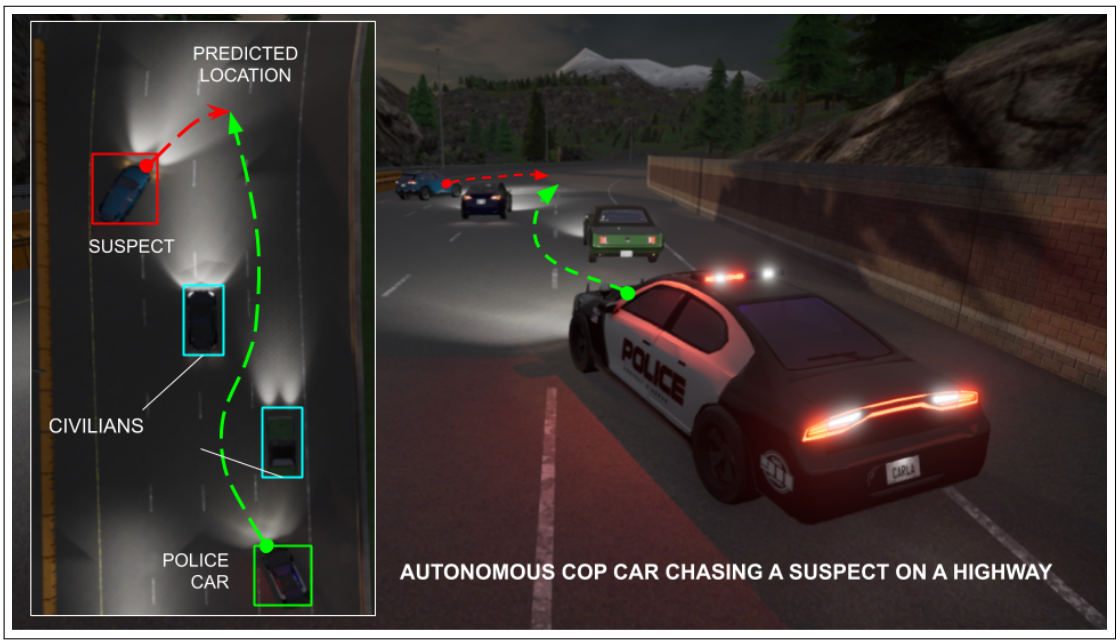


Figure 1.1: A high-speed pursuit of the future, envisioned in CARLA. The green box indicates the law enforcement vehicle (pursuer). The red box indicates the suspect (evader).

To begin with, it is an arduous task to be able to detect and track an actor in complex, ever-changing environments with the use of limited resources. The subsequent pursuit and apprehension of that agent is even more challenging. The aim therefore, is to create a system that provides the pursuer with some form of an advantage over the evading agent so that it is able to pursue and successfully capture it. This tactical advantage can assure robust and consistent performance from the pursuer despite all

the challenges it has to overcome. A prominent example of such an advantage is the ability to predict/estimate the future location of an evader in a given environment. In order to justify this approach, we draw inspiration from nature itself. We take into account how biologically programmed natural predators go about pursuing and hunting their prey. As extrapolated upon in chapter 2, neurological studies have shown that human as well as animal brains have the capacity to form a clear representation of future predictions making use of neurons present in the dorsal anterior cingulate cortex (dACC) of the brain. Predators can take into account the current position and the velocity of their prey and actively predict their future locations. dACC neurons are then used to encode these future positions. We therefore choose to implement a similar strategy that observes the target agent and predicts its future position in the environment. We can also extrapolate to the fact that the more accurate this future estimation is, the better shall be the performance of the pursuer in question. Moreover, there is great benefit to be had from trying to minimize the time it takes for a pursuer to capture an agent that has an adversarial goal of shaking off the pursuit, i.e, the evader. Lower capture times are important because the quicker we finish capturing the evader, the lesser the risk of damages to civilian life and property. It means the society is exposed for a lower amount of time to a factor that would otherwise be considered potentially dangerous, disruptive and unorderly. It can be so derived that the suspect has a lower chance of shaking off the pursuit in a shorter amount of time, since there is now a lower number of opportunities it can take to escape pursuit.

In the work summarized during the course of this thesis, we develop and present a predictive pursuit agent that can be used across different domains (simulators) without the need for explicit retraining i.e, a domain agnostic agent. This agent functions by making use of offline learning to perform online predictions for the pursuit and

capture of another agent in a resource-constrained environment. We set up simulations that make use of mobile robots, one of which is referred to as the Pursuer; the agent tasked with the pursuit and capture of another mobile robot in the same simulated environment, the Evader. The architecture we propose for this task supports any perception sensor that has the capacity to detect the evader and then estimate its location with respect to the agent the sensor is latched on to. Once we have derived a set of observations from the perception module, the pursuer can use the offline model it has previously learned along with the newly obtained observations in order to predict the future positions the evader is likely to be in. The mobile robot in the role of the pursuer then sets course towards the most likely of the future positions of the evading robot.

With the goal of validating this system design and to show that this approach to pursuit is effective, we conducted several experiments in the MORSE [10] simulator. These experiments compare the 'number of steps taken to capture the evader' metric for the proposed method, with the same metric evaluated for conventional estimation methods such as the Kalman Filter [11]. We also show that the proposed method can work across different domains without needing to update the offline motion model for every new domain. In these experiments that so validate the domain agnostic nature of the system, we test the prediction model that was learned offline, without any re-training whatsoever, in another simulator CARLA [9], which shows promising results. The contributions made by the work conducted accumulating into this dissertation are as follows:

- A pursuit-evasion setting that is free from constraints such as the need for a pre-mapped environment or from models that try to learn the behavior of the evader

- An LSTM based motion prediction model with an encoder-decoder architecture that enables the use of an online deterministic policy using an offline motion model of the evader
- Comparative analysis contrasting the results produced by the proposed model to the Kalman Filtering that represents conventional tracking methods and pointing out ways in which the proposed method performs better
- Experiments to establish the domain agnostic nature of the proposed approach to differing environments within a domain, as well as across different domain

Chapter 2

THE NEUROLOGICAL BASIS FOR PREDICTIVE PURSUIT

We search for a pursuer that has the ability to navigate varied and unexplored environments. It should also be capable of robustly going after a suspect in such an environment and apprehend it. We therefore look to find inspiration in nature. There is a wide variety of natural predators whose hunting prowess is greatly acknowledged. From several species of protozoa and bacteria to different kinds of plants such as the Venus Fly Trap, to a variety of fish and insects, predators have gradually evolved over time. The more prominent examples of biologically honed pursuit skills known to us are those of the more ferocious animals from the feline family; lions, leopards, jaguars and tigers to name a few. In order to capture their prey, these predators resort to different kinds of techniques or 'predatory modes'. These pursuit strategies differ in the techniques used by a predator to intercept their target.

One of these techniques is Ambush, where a predator sits and waits to catch their prey by stealth and surprise. This approach is analogous to espionage where a certain area is monitored carefully and the right opportunity is chosen to take action. Another technique known to be used by predators is called Ballistic Interception. This is when the predator observes the movement of its target, predicts its path and then attacks using a ballistic appendage when the target is within reach. Examples of such predators include dragonflies, the archerfish that attacks with a water jet, and chameleons to name a few. This predation technique involves prediction of the future location of the target which reinforces the validity of the pursuit system we propose. However, in the technique described above the pursuer is often static.

There is one other technique several predators make use of, that aligns very closely

with our goal of designing a robust pursuer that consistently performs well. This method where the predator is also a moving component of the environment is known as Pursuit Predation. In pursuit-predation, predators chase an evasive target. It comes down to the relative speeds of both entities involved in the pursuit if the evading entity flees in a straight line. However if the prey executes a maneuver, thereby changing its direction as it flees, the predator has to then predict its new trajectory in real-time in order to be able to intercept it successfully. It also has the need to avoid certain obstacles as they arise during the course of the chase. The possibility of it bearing prior knowledge of the terrain in which the chase takes place is also significantly low. However, even with the odds stacked against them, these predators have built themselves a great reputation for being amongst the best predators on the planet. They are often referred to as 'Apex Predators' i.e, the ones sitting atop the food chain in the animal kingdom as a result of their undeniable skill for pursuit and capture. We therefore dig deeper into this method of pursuit that involves predicting the target's future position, in an effort to understand it from a biological perspective. What gives these predators the ability to temporally track a prey in a given environment? Is it a neurological imperative that endows them with this predatory advantage over other animals?

According to a study conducted by Seng Bum Michael Yoo, Jiaxin Cindy Tu, Steven T. Piantadosi, and Benjamin Yost Hayden [30] that was published in Nature Neuroscience, predictive pursuit has a neural basis to it. This study hypothesizes an improvement in pursuit capabilities brought about by the ability to predict the future position of the prey and then heading towards said position. The task is therefore to search for neurological evidence of mental structures that permit animals to predict the positions of their prey during pursuit. Their experiment involved a laboratory visual pursuit task where three macaques (the subjects) were trained

to perform a joystick controlled pursuit of their prey. During the experiment, the subjects reliably aimed towards their prey's estimated future positions. This implies that the subjects had the ability to generate mental predictions and then use them to guide their behavior. The study also goes on to show that these subjects make use of Newtonian variables such as the prey's position, velocity and acceleration to make these future estimates or predictions. Even more interesting is the fact that the authors were able to identify neurons in the primates' brains that were responsible for tracking the variables describing the state of their prey. Present in the dorsal anterior cingulate cortex or dACC of region of the brain, these neurons multiplexed prediction-relation elemental physical variables such as the prey's current position, its speed and acceleration, with an explicit and clearly distinguishable representation of the future position of the prey.

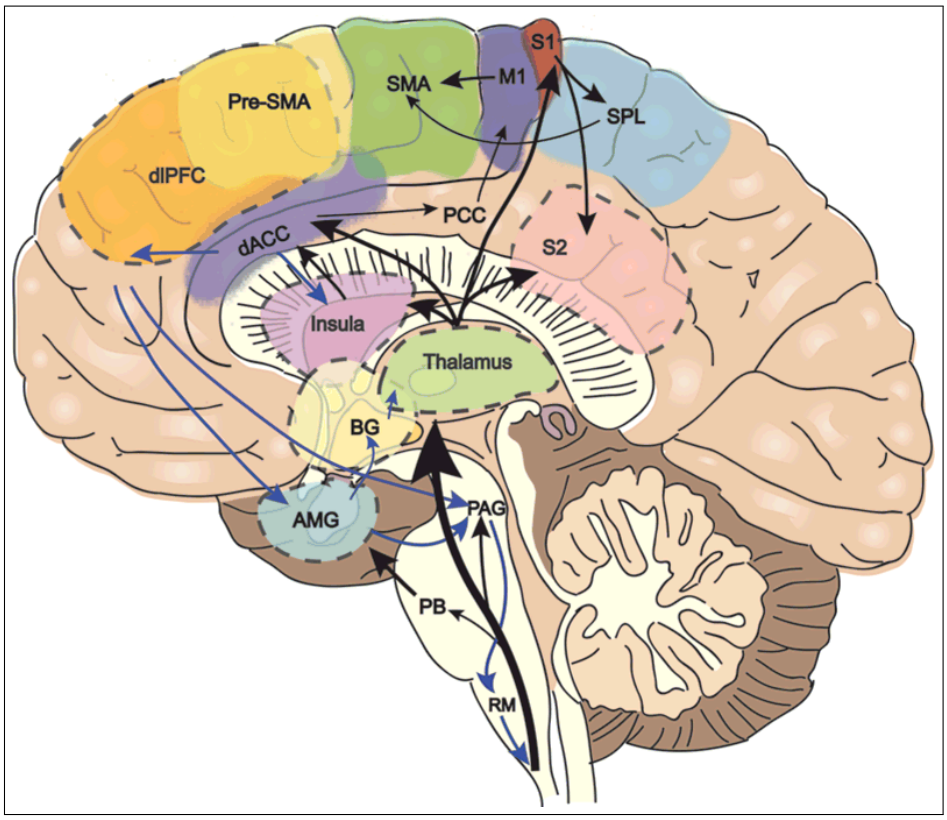


Figure 2.1: An Illustration of the brain anatomy showing different regions of the brain, highlighting the dACC area on the top-left. **Source:** [**21**]

The results from these experiments provide a clear demonstration of the fact that the brain can explicitly represent future estimations making use of the anterior cingulate cortex for such type of cognition.

Many other studies based in biotechnology corroborate the fact that the brains of several excellent predators make use of predictive models that generate future estimations by factoring in the dynamics of their prey. For instance, the paper [16] titled 'Internal models direct dragonfly interception steering' discusses the target reaching behavior of dragonflies involving tracking the angular position of the prey by making predictive rotations of their head. It reinforces that the steering manoeuvres are based on predictive models while they rely on vision to negotiate unexpected move-

ments from the prey. Another paper contributed by Kenneth C Catania [5] explains the predictive model used by a water snake to catch fish. When the prey approaches at a right angle to the snake's head, it anticipates the future position of the fish by striking where the fish would later in its path.

The predatory prowess of many such species in the animal kingdom is well documented and is often a widely acknowledged skillset. We can safely state that this predatorial excellence is fostered by the ability of their brains to use predictive models to generate and store future estimations of the positions of their prey. Their actions as a result are guided by these estimations. We therefore believe a robust pursuer should have similar traits. Thus in our proposed system model, we aim to build a pursuer that makes use of a Long Short-Term Memory based Predictive Model that is analogous to the dACC neurons of the brain. The physical variables from the evader, namely its position is observed by means of a camera mounted on the pursuer, whereas the velocity variable is a derivative of the position over time. Using these Newtonian variables with the predictive model trained offline, we make future estimations of the position of the evader in real-time.

Chapter 3

RELATED WORK

There have been many different times and innumerably many different ways that a game of Pursuit-Evasion has been formulated in the past. Each of these different formulations defines its own set of problems. An attempt at solving these problems then renders a solution, targeting the policies of either the Pursuer or the Evader or that of both the agents involved. Conventionally orchestrated inside of simulators, the quality of a proposed solution to these experimental formulations and its feasibility in real-life scenarios, often depends upon how accurately the problem statement captures the nuances of such scenarios. Therefore, the better a formulation captures the essence of the problem setting to be addressed, i.e the factors critical to it, the more effective the solution. By definition, a Pursuit-Evasion game is set up between two adversarial entities. A Pursuer is an agent with the aim to capture the evader in some sense. The Evader is another entity in the game that either actively (motion policy based directly on the pursuer's behavior to actively deny capture) or passively (as an indirect result of its predefined motion policy) tries to evade capture. In this work, we propose a novel approach for a pursuer that aims to transcend the limitations that arise in conventional pursuit evasion games. As discussed above, our aim with the proposed solution is primarily to try and formulate the Pursuit Evasion game in such a way that it enables us to capture as many real-life intricacies of the problem as possible. It is an arduous task in the sense that every one of these real-life factors we account for in the formulation, limits the implementation of the Pursuit-Evasion game in some way or the other. Therefore, the challenge here is to come up with a solution that works well with high, albeit a realistic number of constraints. To help with the process, we review

and summarize several conventional approaches to Pursuit Evasion. We evaluate these prior approaches for their advantages and discuss the possible limitations that arise with taking these approaches, with a goal of formulating the Pursuit Evasion scenario as pragmatically as possible. This in turn, ensures that the resulting implementation is also more robust and has fewer drawbacks.

There are many different ways in which a Pursuit-evasion game (also known as cops and robbers or graph searching problems) can be set up. They can either involve a singular Pursuer and a single Evader or they can be modelled as Multi-player pursuit-evasion games where there can be multiple pursuers and evaders. Completion criteria in multi-player Pursuit-evasion is defined as the eventual capture of all evaders in the environment. Cases where the evader is not directly visible to the pursuer at the start of the game creates another bifurcation. The problem is then divided into two different parts, namely, search and pursuit. The pursuer has to first locate the evader in the environment and then the game of pursuit evasion can begin. This can also happen intermittently as the Pursuer loses sight of the evader.

Depending upon how the environment where the game is supposed to take place is modelled in the formulation, there are two main types of Pursuit-evasion settings. When the environment is modelled as a graph, the set up is known as 'Discrete Pursuit-Evasion'. This type of modelling was first introduced by Torrence Parsons [28] in the year 1976, where the movement of the agents is constrained by a graph. The agents travel along the nodes of the graph in 'discrete' steps. On the contrary, if the environment is formulated geometrically, it is known as Continuous Pursuit-Evasion. In this type of formulation the environment is modelled geometrically in the sense that it takes the form of the Euclidean Plane or possibly another topological space resembling a Euclidean space. In such games the constraints that can be imposed upon the agents involved, include providing a speed range or limits to acceleration of

the agents thereby restricting their maneuverability. Other variants include 'sweeping problems' ignore the restrictions that mandate the agents in play, i.e, the pursuer and the evader to occupy a certain node in the environment graph. This means that they can also be positioned somewhere along the edges that connect the nodes of the graph. Though several variations of the Pursuit-evasion problem exist, the research in this field is mostly restricted to the Continuous and Discrete Variants.

3.1 Reinforcement Learning

Of the many approaches taken to setting up Pursuit Evasion games in the past, there are those that set it up as Multi-Agent problems. One such approach taken by A.T.Bilgin and E Kadioglu-Urtis of the TOBB University of Economics and Technology [4], that makes use of Reinforcement Learning, discusses this approach to Pursuit Evasion. This particular approach falls under the Discrete Pursuit-Evasion category as the environment is represented as a grid of points i.e it is modelled as a graph covering an arbitrary area. Reinforcement Learning is a method that employs an agent's interactions with the environment is a widely used technique used in the domain of pursuit-evasion. In this paper, instead of a singular pursuer and a singular evader, the problem is formulated as a rivalry between two groups of agents, in which one group attempts to capture the other, while the other group tries to evade. Agents from both the parties involved in Pursuit Evasion make use of the 'Watkin's Q(λ)-learning' algorithm to enable these agents to learn from their interactions with the environment. As the paper aptly describes, Q-learning is a form of reinforcement learning that is an off-policy temporal difference control algorithm. Off-policy learning implies that the policy implemented by the agents during their interaction with the environment is different from the policy the agents want to improve. That is, the

policy deployed for selecting the action an agent needs to take during the learning phase (behavior policy) is different from the policy the agent is trying to learn (target policy). This ensures that the agents can uniformly explore and interact with the environment while learning the target policy and prevents the learned policy from being influenced by the learning which risks it being sub-optimal. The function that decides what action to take next, is referred to as the 'value function'. This function estimates the reward that can be expected under a certain policy, as a score, known as the 'Q-value'. Temporal Difference (TD) learning enables the estimation of these value functions, which would otherwise need to be calculated after waiting for the final reward of the episode and then updating all the state-value pairs accordingly. Thus, in Temporal Difference methods, the final reward is estimated at each step, based on the difference between the predicted rewards for the next time step and the actual rewards for the actions taken thus far. This method is also known as a 'bootstrapping' method because the final estimate is updated to some degree based on an existing estimate and not the actual final reward. In the setup for these experiments, the agents are placed randomly on a closed area and are not provided with any previous knowledge of the map. The peruse(s) is tasked with capturing an evader using a minimum number of moves while the evaders flee for a certain number of rounds. The terminal condition here, or 'capture' is defined by a pursuer occupying the position in the grid where an evader is already located. The episode restarts after the evader in the environment has been captured. A reward structure is defined that decides what actions each of the groups of agents should be punished or rewarded for and a memory known as the state-action matrix is updated according to these rewards.

This setup is designed to mimic real-life scenarios that necessitate the use of multiple pursuers or cases where there are more than one evaders involved in the scene. Using multiple agents is also practical since it enables exploring a relatively

large area. Moreover, it provides redundancy and hence, robust task completion in cases where one of the agents involved malfunctions; contrary to instances where singular agents whose malfunction leads to a failed case. In this paper, two different agent configurations are considered. The first configuration is a normal 1 pursuer - 1 evader setup whereas the second configuration they test consists of 2 pursuers and 1 evader. The paper analyzes and manages the interactions between the 2 pursuers in the second configuration by choosing a concurrent learning approach where each of the two pursuers are independent, i.e, they are solely responsible for their own actions. For the experiments, three different modes are implemented. The first mode has the evader locked to its initial position of the grid. This mode is thus called 'stationary evader' mode. In another mode, the evader performs a random, non-deterministic walk of the environment. This is done to complicate things for the pursuers by increasing obscurity of the evader. This mode is referred to as the 'Random-Walking Evader' mode. Finally, the superiority of the pursuer is challenged in the third mode by enabling the evader to also use the Watkin's Q-learning algorithm where it is heavily punished for being captured and the feedback is sent to its own state-action matrix. Thus the evader now also has a motivation that is adverse to the goal of the pursuer. Experiments are conducted for each of the configurations described above in all different modes possible and the results are summarized.

Three different parameters, namely the learning rate(α), the decay-rate (λ), and the discount factor (γ) are significant to the implementation of the Watkin's Q(λ)-learning algorithm. The learning rate simply identifies what ratio of the new knowledge gained will replace the existing information. If α nears 0, it means the newly learned information has no impact on the knowledge formed by the algorithm. Conversely, an (α) of 1 implies full retention of the new information learned by the agent, discarding its previous knowledge. The decay rate (λ) in this algorithm determines

the extent of the feedback trace, going backwards in time. A λ of 0 necessarily means the agent does not return any feedback to past events. Lastly, the discount factor (γ) decides what impact a future reward shall have on the action an agent decides to take. A low value of this parameter implies an opportunistic agent that values immediate rewards while a higher value of this factor generates an agent that attributes greater importance to a future reward.The pseudo-code for the Watkin's Q(λ)-learning algorithm (also known as Technical Note Q-Learning) is as shown in the image below

Algorithm I: Pseudo code of Watkins's Q(λ) algorithm

Initialize $Q(s, a)$ arbitrarily and $e(s, a) = 0$ for all s, a
Repeat (For each episode):
  Initialize s, a
  Repeat (for each step of episode):
    Take action a, observe r, s'
    Choose a' from s' using ε-greedy policy
    $a^* \leftarrow \text{argmax}_b Q(s', b)$
    $\delta = r + \gamma Q(s', a^*) - Q(s, a)$
    $e(s, a) \leftarrow e(s, a) + 1$
    For all s, a:
      $Q(s, a) \leftarrow Q(s, a) + \alpha\delta e(s, a)$
      If $a' = a^*$, then $e(s, a) \leftarrow \gamma\lambda e(s, a)$
        else $e(s, a) \leftarrow 0$
    $s \leftarrow s'$;
    $a \leftarrow a'$;
  Until s is terminal

Figure 3.1: The pseudo code for Watkin's Q(λ) algorithm as defined in [4]

The results of these experiments indicated that in the scenarios involving a stationary evader, the pursuing agents were able to rapidly discover an optimal path to the evader, in case the above mentioned parameters were optimally set. A trade-off between finding a near-optimal path really quickly and an optimal path relatively slowly was also observed. The scenario where a random-walking evader was involved proved to be an area of emphatic successes for the evader. The pursuer(s) in this case were unable to derive useful information as the movement of the evader was completely uncertain. Finally in cases where both the agents were able to make use of the Q-Learning algorithm, the outcome depended on their respective learning rates. The setup with 2 pursuers and 1 evader exhibited effects of varied geographical distributions and increased robustness, compared to the more conventional 1 pursuer - 1 evader setting. Another important observation from the results presented in the paper, is that even in cases where there is no explicit cooperation between the members of a pursuing team, simply the numbers advantage over the adversary is of a high significance as far as the pursuit-evasion problem is concerned. Other works such as [25], [29], [32] and [6] discuss the use of reinforcement learning approaches in multi-agent scenarios.

A few key takeaways from the above discussion of the research issues help us formulate our own pursuit-evasion scenario and they are as follows:

- A random-walking evader is the most difficult adversary as it introduces a huge amount of uncertainty to the pursuit-evasion game. Being able to consistently capture an eccentric evader requires a really powerful and robust pursuit strategy.

- The pursuit-evasion scenario we aim to set up consists of an environment that is modelled geometrically. Thus, physical variables such as velocity and accel-

eration also need to be accounted for in the experimental setup. Therefore, a discrete learning algorithm such as the Watkin's algorithm used here is not a suitable choice for the target scenario. Based on their advantages and disadvantages, a choice needs to be made between using a Deep Reinforcement Learning Model or not using a Reinforcement Learning algorithm at all.

- One might additionally argue that setting up a single pursuer - single evader experiment is sufficient to infer the effectiveness of the pursuit strategy learned, since concurrent learning in multi-player scenarios simply amplifies the effects of the difference in the number of members in adversarial teams

3.2 Differential Games

It is common to approach the formulation of pursuit-evasion games by imposing some restrictions on the adversary's actions by making certain assumptions regarding their behavior. This makes it easier to devise an optimal strategy for the pursuer since these restrictions limit the possible behaviors of the evader and vice versa. This however, does not address the possibility of an intelligent opponent that does not abide by these limited set of actions. Thus, the goal of being able to come up with optimal strategies for the pursuer that account for even the worst possible behavior of the evader, led to the emergence of Differential Game Theory. The creation of strategies that are optimal for either of the teams involved in pursuit-evasion (the pursuer and the evader teams) regardless of the actual policies implemented by the adversary, is the purpose of formulating pursuit evasion scenarios in differential game settings.

Preliminary development of differential games stems from the works published by R. Isaacs [27] in the year 1965, in which he introduced the idea of posing problems

in a framework based on dynamic game theory. He referred to this paradigm as 'Differential Games' which made use of the principles of game theory, calculus and control theory to solve problems between multiple players/agents. He introduced several critical mathematical constructs such as dispersal, universal and equivocal surfaces that described the optimal flow field in differential dynamic games. The goal of differential dynamic programming is to come up with what are known as 'saddle point strategies' for all the agents involved in the game. A saddle point, or minimax point is a point of the surface of a graph of a function, where the slopes i.e, the derivatives in orthogonal directions are all zero. The surface typically used to describe a two-dimensional surface is one that curves up in one of those directions and curves down in the other which resembles the shape of a riding saddle. Hence the name 'saddle' point. In a zero-sum game between two adversarial entities (or two such opposing teams) the equilibrium point is the saddle point. In terms of Differential Games, saddle point strategies are signified by the optimal strategies that are to be devised ultimately implemented by agents from either of the adversarial teams.

Several classical differential games, although often discussed recreationally, can be made use of, as model problems that are pivotal in deriving the actual mathematics used in several real-life applications of Differential Games. An example of such a differential game is the 'Homicidal Chauffeur Problem' [17]. Proposed by Isaacs this famous classical problem consists of a holonomic pedestrian that is highly maneuverable, but slow whose adversary is the driver (chauffeur) of a motor vehicle that is fast but has now maneuverability compared to that of the pedestrian. The aim of the driver is to run over the pedestrian with their vehicle and the aim of the pedestrian conversely, is to elude the car and prevent themselves from being run over. Hence the name 'Homicidal Chauffeur'. Several questions thus need to be addressed as a result of the way this problem has been set up. For instance, what strategy can the

driver of the less maneuverable car use, in order to guarantee them running over the pedestrian? Also, strategy should the pedestrian, i.e the evader deploy to be able to elude the chauffeur indefinitely? If the capture of the evader, or here the metaphorical demise of the pedestrian is guaranteed, what can the chauffeur do to minimize the time it takes to run the pedestrian over? Saddle-point strategies for both the pedestrian and the homicidal driver can thus be derived by making use of the agents' dynamics models. This problem, depending on the surface where the pursuit-evasion setting occurs (topological variations of land, air) can be used for different applications. It is often used as an unclassified proxy to missile defense and other targeting problems by scientists to enable them being published without incurring any serious security implications.

Another classical variation of the differential game is the Two Cars Differential Game [26] where each of the players is in control of a car with minimum turning radius. Both these players are engaged in an adversarial game of pursuit-evasion. The relative turning radius of these cars also spawns several new variations to the Two Cars Differential Game. The dynamics models of the cars are known beforehand and various properties of the game, such as the regions of capture, regions of escape and the barrier surfaces between those regions also add to the game definition. Using the parameters in addition to the dynamics models of the agents, the optimal policies for the parties involved are derived. Several details such as the effect of differing the agent's speed, capture radius and other maneuverability constraints also make these games more complicated. Similar to the Homicidal Chauffeur problem, this classical model can also be interpreted as an aerial problem by modelling the state space as a three-dimensional plot. Figure 3.2 shows an example of the two-dimensional motion models of the two cars that engage in a differential game.

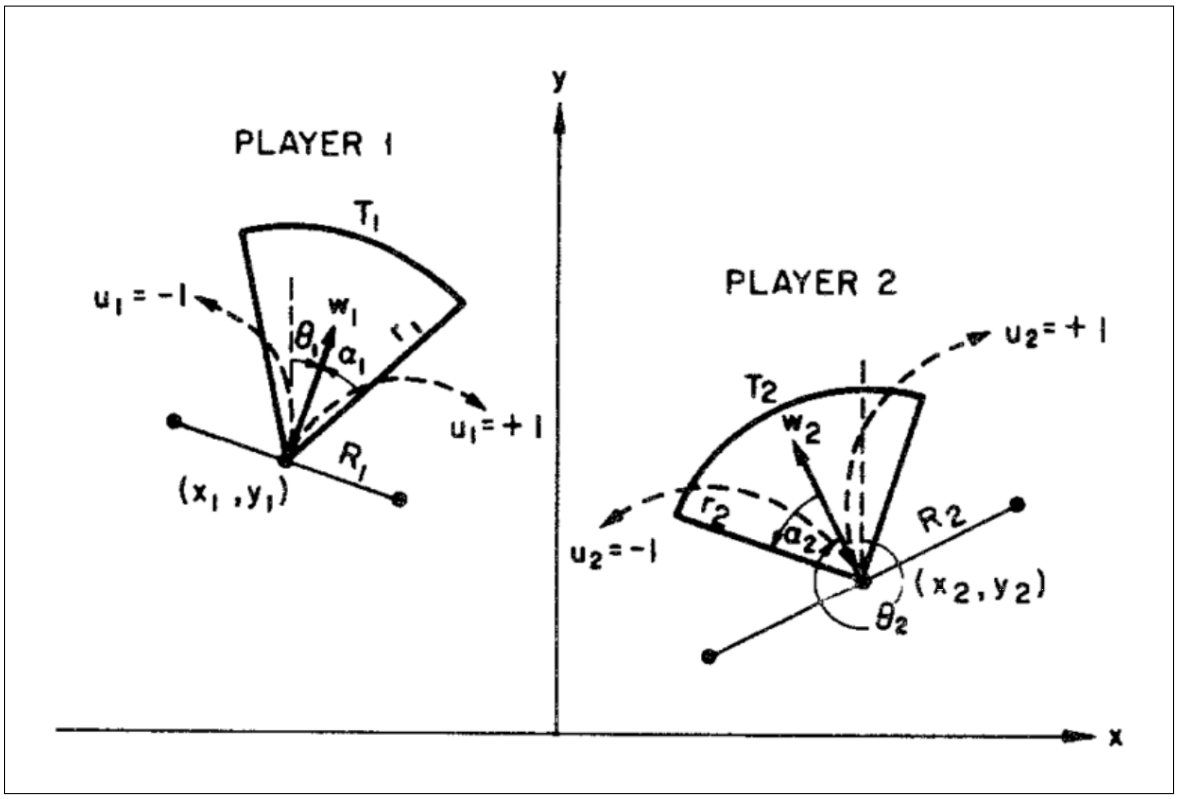


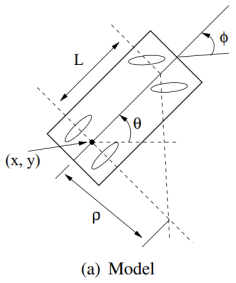
Figure 3.2: 2 - dimensional motion models of cars where $R1$ and $R2$ are the minimum turning radii and $u1$ and $u2$ describe the max rate turn curves. Vectors $w1$ and $w2$ describe the velocities of the respective cars.

As discussed by authors Isaac E. Weintraub, Meir Pachter, and Eloy Garcia in [26], seminal contributions to optimal control and static/dynamic games that deserve recognition include works by scientists such as Richard E. Bellman, Leonard D. Berkovitz, David H. Blackwell, Melvin Dresher, Wendell H. Flemming, and John F. Nash as well as their Soviet counterparts N. Krasovskii, A. Melikyan, L. S. Pontryagin, and A. I. Subbotin. Further work in the field of dynamic games was inspired by the first conference that was dedicated exclusively to this topic, which was called the 'First International Conference on the Theory and Applications of Differential

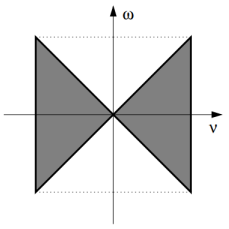
Games', organized in Amherst, MA between September 29 and October 1 1969. This preliminary initiative resulted in some pivotal work involving pursuit-evasion games in the more recent past. Provided the initial configurations of the Pursuer and the Evader, the differential games provide the necessary and sufficient conditions for the capture of the evader.

Papers such as the one by authors Ubaldo Ruiz, Rafael Murrieta-Cid, and Jose Luis Marroquin [20] considers the problem of capturing an evader with the ability to move in all directions, using a pursuer that uses a Differential Drive Robot (DDR) in an environment free of obstacles. Here the goal of the evader is to stay out of the pursuer's range of capture for as long as possible, whereas the aim of the pursuing agent is to capture the evader as soon as possible. This means the strategies developed as a solution to this problem results in a time-optimal strategy for the Pursuer. Making use of the differential drive robot for the pursuer means that it is much more maneuverable than the pursuer which in turn is modelled after a regular car. This makes the setting of this problem analogous to the Homicidal Chauffeur problem discussed above. Therefore in the essence of making the less maneuverable agent faster and the more agile agent slower, the DDR is allowed a higher maximum velocity than the evader. The DDR can change direction at a rate inversely proportional to its translational velocity. Acceleration bounds are not considered, making this a purely kinematic problem. The goal is to find optimal strategies for both agents that are in Nash Equilibrium. This means that even if one agent decides to change their policy unilaterally, they won't benefit from this new policy any more than they would, following the optimally derived strategy. Provided with the kinematics models of both the agents involved as shown in Figure 3.2, and the definition of the state space representation being considered, optimal pursuit and evasion policies for the corresponding agents are then derived making use of Isaac's Methodology (IM). The IM is an ex-

tension of the Haminton-Jacobi-Bellman (HJB) [12] equation, that provides sufficient conditions that ensure the existence of saddle-point equilibrium for strategies of both the parties involved in the pursuit-evasion game. Another paper by Bhattacharya [3] also sets up the problem as a game-theoretic visibility-based pursuit-evasion game.

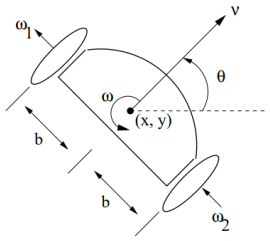


(a) Model

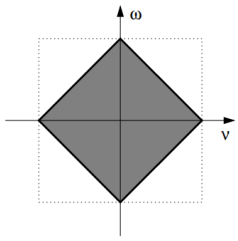


(b) Control space

Figure 3.3: Car-like Robot



(a) Model



(b) Control space

Figure 3.4: DDR

The study of pursuit-evasion scenarios set up as differential games in light of the classical examples of differential problems, such as the Homicidal Chauffeur and the Two Car problem, helps us build the foundation to a robust pursuer that we aim to achieve by means of this work. The important things to notice from these implementations follows:

- Setting up the pursuer and the evader in an environment modelled as a Differential Game is an entirely deterministic implementation. This methodology thus has the advantage of being lightweight in terms execution times and can be relied upon to produce the optimal strategies for both the agents in real-time.

- Differential Games can be set up in a continuous space, using continuous time-steps, which is a suitable quality for such implementations keeping in mind, the robust, all-terrain pursuer we wish to build in this project.

- However, factoring in more physical constraints, such as upper thresholds to acceleration and acknowledging the presence of obstacles makes modifying the implementation as a differential game, immensely more complex and challenging.

- Moreover, one of the greatest challenges in formulating a pursuit-evasion game as a Differential Game, lies in the fact that the dynamics models of both the agents involved need to be known in advance. It is not always possible to obtain the kinematics model of the evader in advance, especially in spontaneous scenarios such as the one described at the start of chapter 1, where the premise dictates that we possess minimal knowledge of the evader.

3.3 Graph Theory

As discussed previously in at the start of chapter 3, pursuit-evasion games can be set up in either a probabilistic manner [24], [14] or as graph problems [13], [15]. Graph-based pursuit problems are typically modelled as discrete problems where the players traverse the nodes of a connected graph, making use of the edges between the nodes. These agents are often adversarial in case of pursuit-evasion. Graph formulations have predominantly been used to solve Search Problems. In this case the party being searched for can also sometimes be immobile. For instance, the search for a hiker injured in the woods can be modelled as a graph problem, with the area covered by the forest limiting the length of the graph. The hiker here, since injured, is stranded at a fixed location. The goal of the rescue team is to parse the nodes of the graph to locate the hiker to ultimately be able to rescue them. In another example of the search problem however, the other agent can also be mobile. For example a spelunker that's lost in a cave and needs to be rescued. In this scenario,

the lost explorer might actively try to roam around the cave to find the exit even while the search team is looking for him. This makes the job of locating him much more difficult, as the lost spelunker might reoccupy after a certain amount of time, areas that the rescue team has already marked as explored. Amongst a variety of solutions, questions such as finding the minimum number of workers that are needed to rescue the lost cave explorer can also be solved making use of graph theory. The cave can be modelled as a finite connected graph in which the lost person and the searchers move continuously. The person being searched for, might also be evasive in nature in some cases. The above example involving the injured hiker shall be analogous to pursuing an intelligent evader, if the hiker in the scenario is replaced with an escaped convict, being looked for by the cops. The convict actively tries to escape capture, acting as an adversary to the cops whose actions are aimed at locating and capturing him. The algorithms used to solve these kinds of situations hence fall into the pursuit-evasion category. These solutions need to account for the movement of the agent to be captured or located, in addition to simply conducting an organized search of the nodes of the environment modelled grid. A typical structure to these algorithms is defined by agents that make decisions based on the value of an entity (usually a variable) they track. The agents traverse the graph with the aim of either maximizing or minimizing the value of this variable. Every node is assigned a value that is based on the nature of the problem to be solved and it either adds to, or takes away from the value of the variable being tracked once an agent traverses to that particular node.

The paper titled 'Tree Search Techniques for Minimizing Detectability and Maximizing Visibility' [31] describes such a problem where the search environment is modelled as a grid where each cell has a reward value associated with it. The paper begins by discussing general planning problems where the goal is to plan a path for

an agent that enables it to cover the maximum possible area, restricted by a time limit. This problem is analogous to the classical Watchman Route Problem which is an optimization problem, provided with a map of the area, a watchman is tasked with guarding an entire area that also contains obstacles. The watchman hence needs to discover the best path to traverse the discrete environment, peeking behind every corner while it does so. The aim of the planning problem can also be to minimize the time needed to cover the entire environment as a whole. This problem resembles the Art Gallery Problem where the task is to determine the minimum number of cameras required to maintain sight of all the points in a polygonal environment.

In this paper, scenarios that contain another adversarial agent in addition to the one covering the span of the environment, are discussed. The adversarial agent or a guard, is tasked with locating the agent that's traversing the environment, while also trying to be undetected by the guard. All the points in the map previously seen by the agent are considered as covered and those currently visible to the guard constitute its area of surveillance. Similar to the paper discussed under the Differential Games variety of pursuit-evasion in section 3.2, the problem in this particular paper is also set up as a Zero-Sum game. The agent that explores the environment, gets positive rewards for maximizing its visibility and gets penalized for being detected by the guard. It is tasked with different missions. In case of an exploratory mission, the task is to explore the environment and the reward is a function of the number of previously unseen cells, now visible from the agent's current position. Another type of mission is where the goal of the agent is to reach a target location without being spotted by the guard. Similar to the previous mission, the agent receives a negative reward for being in the area visible to the guard. In addition to these penalties, the agent is also rewarded incrementally for getting closer to the target location. The closer it is to the target, the higher the reward. Thus the reward can be described as

a function of the distance of the agent to its goal location. This setting is a type of visibility-based pursuit-evasion. However, it is different from classical pursuit-evasion game since the goal of the evader (agent) here is not only to evade the guard but also to either increase its visibility or to reach a goal location simultaneously.

The solution can be implemented using two types of searches; a Minimax Tree Search and a Monte-Carlo Tree Search. The agent that explores the environment or travels to a goal location is termed as the MAX player as it is trying to maximize the reward it obtains. The guard on the other hand is referred to as the MIN player as it aims to minimize the total reward. Although both agents move simultaneously, each step in the game is modelled as a turn-based step. Minimax Tree search is commonly used to solve zero-sum problems such as this one. This continues to a total of T time steps which is determined by the time budget allocated to the agent's task. The tree constructed for Minimax Search as shown in Figure 3.5 has a Root node, containing the initial positions of the agent and the guard, a MAX level and a MIN level. Every node of this tree stores information such as the position of the agent, the guard's position, the area visible to the agent in its path from the root node to the node it's currently on, and the number of times it was detected by the guard on the said path. In the MAX level, branches are created for neighbors of the agent's position in the previous level (either root node or MIN level). The MAX level does not track the position of the guard. Similarly, the MIN level expands the tree for each neighbor of the guard's position in it's parent level (which is always a MAX level) and the agent's positions aren't updated in the nodes at the MIN level. Terminal Nodes are the leaf nodes from when the Minimax Tree has been fully generated and they are always MIN nodes. The reward values are backpropagated from these terminal nodes all the way back to the root node. As with the minimax policy, an action is chosen for either of the agents that maximizes and minimizes the reward that is propagated

back to the MAX and MIN level nodes respectively. The Monte-Carlo Tree Search (MCTS) method on the other hand is a more time and space optimal algorithm. Instead of expanding on each and every neighbor of the parent node in the previous level of the tree, the tree is now expanded by carefully selecting one of those nodes to expand. This selection depends on the current estimated value of the node in question. This estimate is found by means of rollouts. The rollout is analogous to simulating an arbitrary game from each of the nodes to be selected from and then the node with the best estimate is chosen for expansion. It is shown in the paper that the MCTS are by approximation, one order of computational time faster than their naive Minimax counterparts. Both these algorithms have thoroughly been explained in the paper. Another interesting technique used to reduce the size of both the minimax tree and the MCTS is called 'Pruning'. Pruning a node means that the particular node won't be subject to expansion. This saves the time that would otherwise be spent calculating all the successors of the pruned node. Different strategies such as the classical alpha-beta pruning have been used for purposes of pruning in the paper.

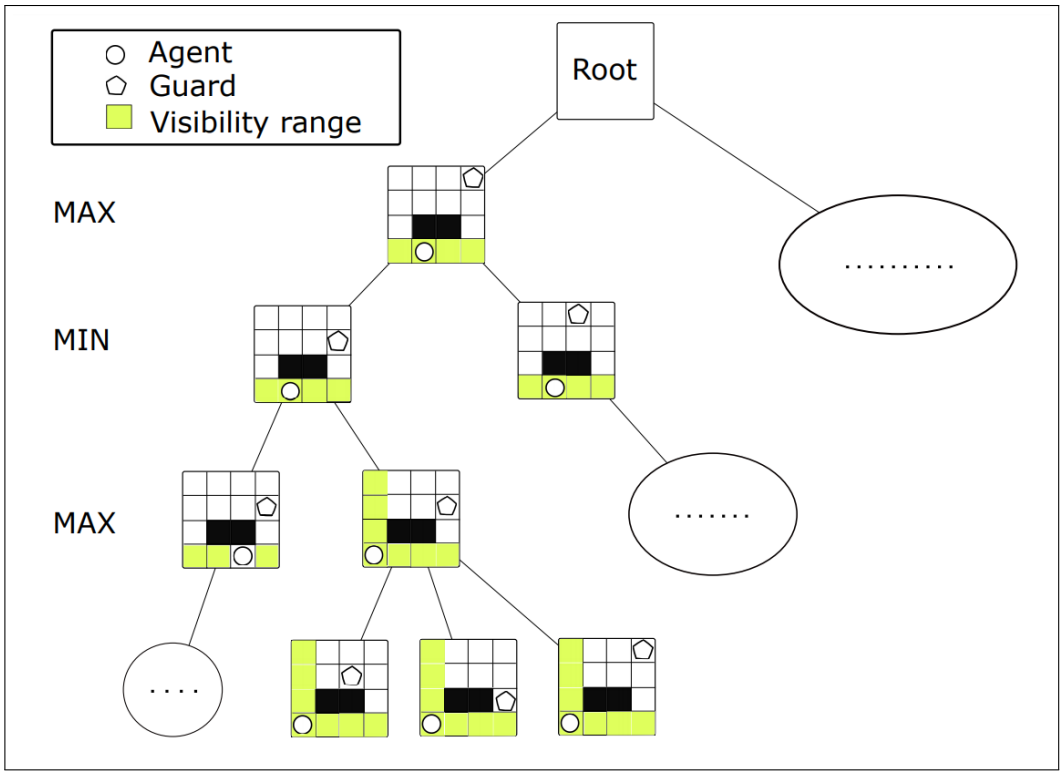


Figure 3.5: MINIMAX Tree for Graph-based Pursuit-Evasion

This paper thus introduces another form of visibility-based pursuit-evasion problems, where the problem of maximizing visibility while minimizing detection is presented. The environment consists of two agents, an agent in charge of the exploration and a guard that is tasked with the surveillance of the environment. Both the agents have knowledge of the map they are in. The problem is solved using Minimax and Monte-Carlo Tree Search techniques aided by pruning to make the computation of the trees time optimal by reducing their size. However, open environments are hard to discretize spatially as well as temporally. The MCTS or the MINIMAX tree formed would be indeterminable in an unmapped environment where there is no spatial constraint on area habitable by the agents involved in pursuit-evasion. The hard limitation imposed by the requirement for the environment to be reasonably small and pre-mapped, prevents the graphical formulation of pursuit-evasion from being the ideal solution to the problem statement we propose.

3.4 Emergent Learning

Another really interesting take on pursuit-evasion games which has opened up new avenues for research in game theory is the work on Emergent Learning. To cite a particular instance of it, the work presented by Bowen Baker, Igor Mordatch and co from OpenAI and Google Brain in the paper titled 'Emergent Tool Use From Multi-Agent Auto Curricula' [2] presents encouraging results and revolutionary insights into the future possibilities and the scope of deep learning enabled pursuit-evasion. This work describes how intelligent artificial agents can perform a variety of complex tasks, specially those that involve these agents being able to sense and interact with objects in the real world. Instead of designing a reward structure and training the agents using reinforcement learning to perform a particular task they need to perform, indirect exploration techniques are utilized. This preserves the ability of these agents to perform highly complex tasks, the reward structures for which are either too time-consuming and too task-specific, or simply impossible to design. Conventionally used undirected exploration methods scale poorly to the vast complexity of open environments. Thus, here the phenomenon where multiple agents when subjected to an environment develop implicit strategies by creating new tasks for each other, is utilized. This phenomenon that creates these implicit 'auto curricula' leveraging the other competing agents in the environment, is evidently abundant in the process of evolution, where organisms co-evolved and competed with each other as a part of the process of natural selection (Dawkins and Krebs, 1979). In this process when a new mutation appears or if some species successfully resorts to a 'curriculum', it creates a pressure on the other species to develop their own competing curricula to mutate and survive. The species that are successful in doing so survive as a result and the process keeps on repeating. The only prominent drawback to this methodology that needs

to be discussed in tandem with the results that it has been able to provide, is the sheer amount of computational firepower needed to be able to perform experiments of this magnitude. 'Episodes' of Reinforcement Learning algorithms need to run for millions if not billions of iterations for the agents to be able to learn new skills as part of emergent learning. However, parallelization is a viable option depending on the availability of capable hardware. Moreover, the rate at which the technology stack associated with acceleration in GPUs and Deep Learning is evolving implies that in the near future, similar tasks with unusually high processing requirements shall perhaps be considered to be of the run-of-the-mill, trivial kind. This approach mimics evolution and the skills learned by all sentient life-forms as a part of the evolutionary process that enabled their mutation. Evolution is a process that is spread over multiple billion years. The realization that comes from comparing that span of time to the unbelievably miniscule amount of time it takes for these agents to learn emergent skills shall, further encourage the utilization and advancement of deep reinforcement learning techniques that pioneer the establishment of new and more intelligent solutions to pursuit-evasion scenarios.

The experiment discussed in this paper is set up as a game of hide-and-seek. This is analogous to a pursuit-evasion setting which also contains agents with adversarial mindsets. Simply put, the goal of the hiders is to use the environment to hide and prevent the seekers from ever being able to find them. This game is designed as a Reinforcement Learning scenario where the seekers/pursuers are incentivised to find these hiders/evaders. Provided items in the environment that can be interacted with, these agents, over the course of millions of such Reinforcement Learning episodes, learnt to use these objects to their advantage. These techniques they learnt are referred to as 'Emergent Skills' as they emerge after prolonged learning and multiple iterations in the environment consisting of interactable objects and other competitive

as well as friendly agents. These agents thus learn an implicit 'auto-curriculum' by playing against present as well as past versions of themselves in parallel, utilizing a technique that is called 'self-play' for faster and more thorough learning. It has also been observed as a part of this research that these techniques scale well to more complex scenarios situated in large environments because of the leverage provided by competing agents in policy learning as opposed to conventionally utilized self-targeted rewards and learning strategies. Emergent Tool learning also implies learning more human-relevant skills (the use of tools) compared to the learning based around reward structures that provide an agent with entirely intrinsic motivation. As a part of this work, open-source code and environments are also contributed to encourage further research in the area. The setup for this experiment consists of agents that are competing in a two-team hide-and-seek game inside a physics-based environment. Hiding and Seeking is all line-of-sight based, i.e visual hide-and-seek where players have to be visible to their counterparts to be pursued. Objects that these agents can interact with are randomly spread throughout the environment. These are no explicit incentives provided by the reward structure for interacting with these objects to either of the teams. The only reinforcement being delivered is for the game of hide-and-seek. The rewards given are team-based. All hiders have to be hidden from the seekers for the hider team to be rewarded positively. Even if one member from the team is visible to the seekers, the entire team gets a negative reward. The seekers in the game get the opposite rewards. If they aren't able to see even a single hider, they get penalized. Space constraints are enforced by penalizing the agents by a huge negative value if they stray too far away from the playing field. In an episode, before the game of hide-and-seek actually begins, a certain amount of time is allotted to what is known as the 'preparation phase'. During this period, all the seekers are immobilized so that the hiders can either run away or change their immediate surroundings. Simulation is

done inside the MUJOCO Physics Engine and it is populated with a varying number of hiders, seekers and movable boxes/ramps. The agents have been equipped with range sensors similar to LIDARs and have a cone of 135 degrees in front of them that defines their field-of-view. The agents that act as hiders and seekers are modelled as spheres. The move by applying forces in the x and y directions and turn by adding a torque along the z-axis. Each of these agents can choose to perform one of three different actions for each time step. These actions are all binary, i.e on or off and can be applied to objects inside a certain radius from the agent that wants to perform the action. Agents can 'grab' objects which binds the objects to the agents and can then move around along with the agent. They can lock objects to the current positions they are in which can only be unlocked by members belonging to the same team as the agent who locked them. The agents learn via self-play using two separate networks, namely an action network and a critic network. As the agents are allowed to train for more and more number of episodes, different strategies emerge as episodes of hide-and-seek (of the order of hundred of millions) are played. There are as many as six of these different emergent behaviors observed over a course of around 500 million episodes of deep reinforcement learning using multiple agents.

In the beginning the hiders and seekers played the basic game by learning to chase and run away. These are most crude behaviors that were an obvious consequence of the deep learning reward structure incentivising the game of hide-and-seek. After about 25 million episodes with this policy, the hiders learn to use the movable objects inside the environment to their advantage. They modify the environment around them to create shelters by locking in place the movable objects, in an attempt to hide themselves from the seekers more permanently. This development creates a pressure on the seeking team that results in them improvising their strategy after another 75 million iterations of hide-and-seek. They overcome the defense strategies exhibited

by the hiders by learning to move and use ramps to jump inside the shelters built by them. With the pressure now being back on the hiders to not get penalized, they learn to take away the tools used by the seekers. They did this by moving the ramps to the edges of the play area and then locking them there.

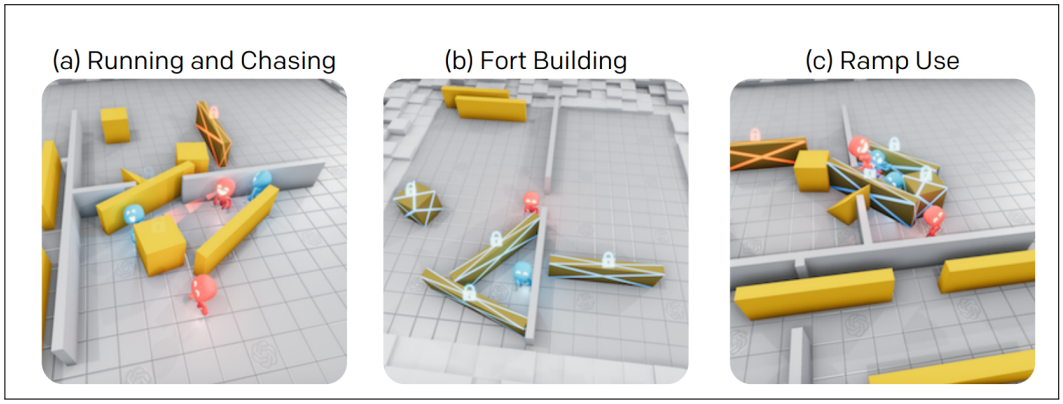


Figure 3.6: The first 3 emergent skills learnt from the curriculum, namely Running and Chasing, Fort Building and Ramp Use respectively

As seekers are on the opposite team, they couldn't unlock the ramps, rendering them useless. Although this was believed to be the last learnable skill that emerged from the reinforcement learning schedule, after a total of about 380 million episodes of training, the seekers were found to have learned another new behavior. They had learned to bring boxes near the ramps locked by the hiders. They then climbed on top of these boxes using the ramps, 'grab' the boxes, and then 'surf' these boxes to the hider's shelters. They could accomplish this by applying force in the direction they want to move in, while being attached to the boxes.

Another unexpected behavior that was learned was by the defenders or the hiders in response to the box-surfing learned by the seeker team. They now also learned to lock all boxes in place, in addition to the ramps, before starting to build their shelters. These are all the behaviors that emerged during all stages of learning. The paper also discusses the effects of the number of players present in either of the teams,

on the learned strategies. Different supervised cognition and memory tasks were also experimented with that involved counting and manipulation.

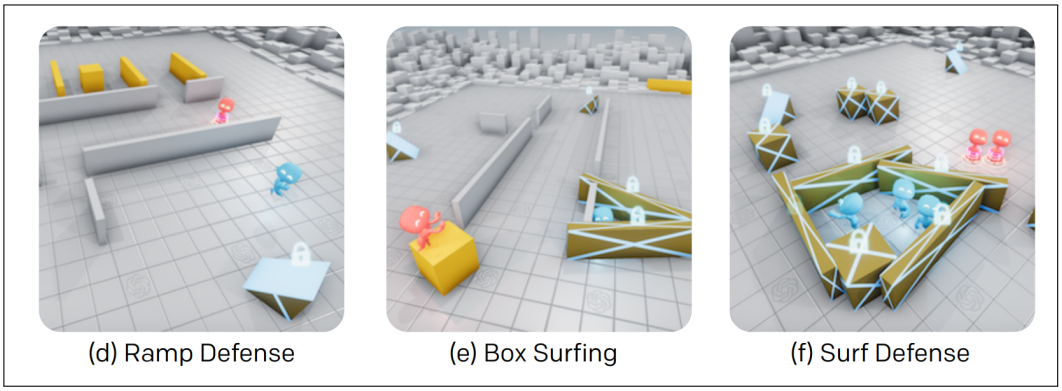


Figure 3.7: The latter of the skills emerging from the curriculum after 75 million training episodes

After having studied this paper, there are a few things of note.

- Primarily, the computational cost of algorithms such as the one presented in the paper, are incredibly high. Tasks of this magnitude often require a number of CPU/GPU workers of the order of 2^8 to 2^{15}. The current RL infrastructure at the academic level and for most of the industry cannot scale to match the stress induced by the training of these gigantic networks.

- Acceleration of such algorithms with the hardware constraints that exist, is a challenging task. The computational load from the task to be executed also depends on several other factors such as the degree of realism to be availed from the simulator in use.

- It is an extremely challenging task to incorporate the entropy in environments that are open to be explored and haven't been previously mapped. In the algorithm described in the paper, the play area had to be constrained by penalizing

the agents for going out of it, in order for them to be able learn emergent skills by making hundreds of millions of iterations of the same environment. It is not practical to implement such a solution unless the agents can be provided with the knowledge of all the objects they might come across in the environment and how to interact with them. Even if they possessed the knowledge in question, the number of episodes it would take for the agents to learn new behaviors associated with these objects is incomprehensibly high.

Chapter 4

THE SYSTEM MODEL

We set up our experiment scenario in the form of a Pursuit-Evasion game. Pursuit Evasion games, as described previously, are problems historically formulated within the intersection of the domains of Mathematics and Computer Science. There are several variants to how these problems are defined and an even widely varied set of solutions engineered to solve them. By general definition, a pursuer or a group of pursuers is tasked with capturing an evader or a group of evaders. The pursuer implements different strategies with the end goal of capturing the evader while the evader either behaves randomly or actively evades the pursuer depending upon the overall aim of the experiment.

In this particular setup built inside of the MORSE simulator, we have two agents, namely the Pursuer and the Evader that are modelled as 4-wheeled vehicles. Both these agents possess the ability to navigate the environment by making use of their individual 'Navigation Stacks'. Since we want our agents to negotiate an unmapped area based on the premise of the problem we aim to solve, these navigation stacks help the agent travel from point A in the environment, to the destination provided to it, i.e point B. For the purposes of avoiding obstacles in the route, they are equipped with a LIDAR sensor each and host an algorithm that uses the data generated by these LIDARs to help the agents take an evasive course when obstacles are encountered.

The evader moves randomly in the environment. This is achieved by sending the evader's navigation stack, random locations in the environment as goal locations. We choose to set up the evader in such a manner, because it is our belief that a random evader brings the most amount of uncertainty to the pursuit-evasion game. This

makes the task of capturing said evader really difficult for the pursuer. Success in a challenging scenario such as the one created by the arbitrary motile evader attests to the robustness of the solution executed by the pursuing agent.

In addition to the navigation stack equipped by both the agents, the pursuer also has access to a Prediction module. This module comprises a camera that tries to detect the evader in the frames it captures and processes this data to determine the location of the evader in the environment, relative to itself. These locations are also stacked and sent out to a Prediction module that forms the core of the solution we propose. This prediction LSTM estimates the locations the evader might occupy inside the environment in the future. Depending on their availability the pursuer makes a choice between the current and the predicted locations of the evader. This location hence chosen, is sent as a goal location to the pursuer's navigation stack. An episode of this pursuit-evasion scenario ends either the capture of the evader or with the evader getting away from the pursuer.

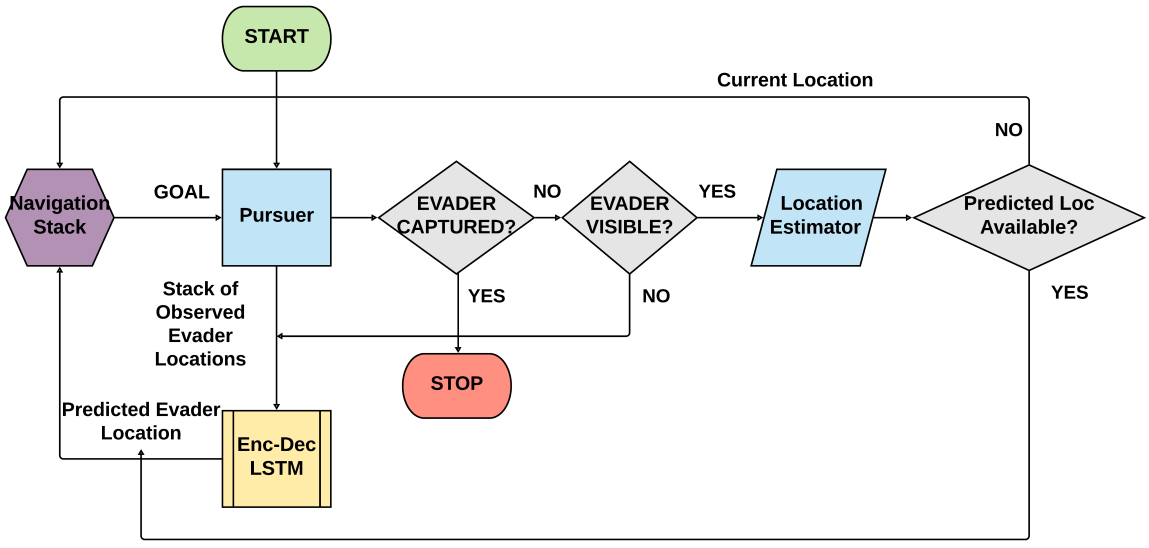


Figure 4.1: Overview of the System Model for Active Pursuit

4.1.1 The LSTM Architecture for Prediction

For predicting the future state of the evader in a geometrically defined environment, the proposed pursuer makes use of a Prediction Module. This module uses the camera on board the Pursuer as input, and as output predicts the future location of the evader, with respect to the pursuer's current location. The processing phase of this module involves the use of an Encoder-Decoder LSTM. This LSTM is fed the consecutive locations of the evader as observed by the pursuer in the past as a sequence. It then predicts another sequence of locations that consists of the evader's future locations. These predictions form the core of the system model we propose as a solution to the game of Pursuit Evasion so set up.

Sequence to Sequence (also known as seq2seq) is an architecture that converts one sequence to another. The principal components of any Sequence to Sequence architecture are an encoder network and a decoder network. The task of the encoder is to convert the input sequence into what is called a 'hidden vector' which is a representation of the input vector. This hidden vector essentially encodes the input sequence and the context it appeared in. The decoder is a complementary network that reverses the same process, i.e, converts the vector representation back to the item and its context. Seq2seq models are commonly constructed using either a Recurrent Neural Network (RNN) or more frequently Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU) to tackle the vanishing gradient problem.

For the purposes of this project, we make use of the Encoder-Decoder LSTM. It is a recurrent neural network that is designed to solve sequence to sequence problems. As described above, it derives its name from the two networks namely, the encoder

and the decoder working in tandem. The Encoder-Decoder LSTM was developed primarily for natural language processing problems and demonstrated state-of-the-art performances in an application of text translation called statistical machine translation [7]. They are also capable of producing similar results in the areas such as time-series forecasting since these problems are analogous to seq2seq problems. Other papers [22] also discuss the use of multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of fixed dimensionality and then another deep LSTM that decodes the target sequence from the main vector.

During the training phase, we supply the Encoder-Decoder with sequences containing the past locations of the evader, as observed by the pursuing agent. The corresponding outputs to the sequences are the actual locations of the evader recorded in the environment, that act as ground truth data. As a result, the LSTM model learns to predict the future locations of the evader as a function of the locations it has already travelled to in the past. Thus, when provided with a set of previously unseen locations, we are able to produce a sequence containing a set of probable future locations. A generic schematic for an Encoder-Decoder LSTM is as shown in Figure 4.2.

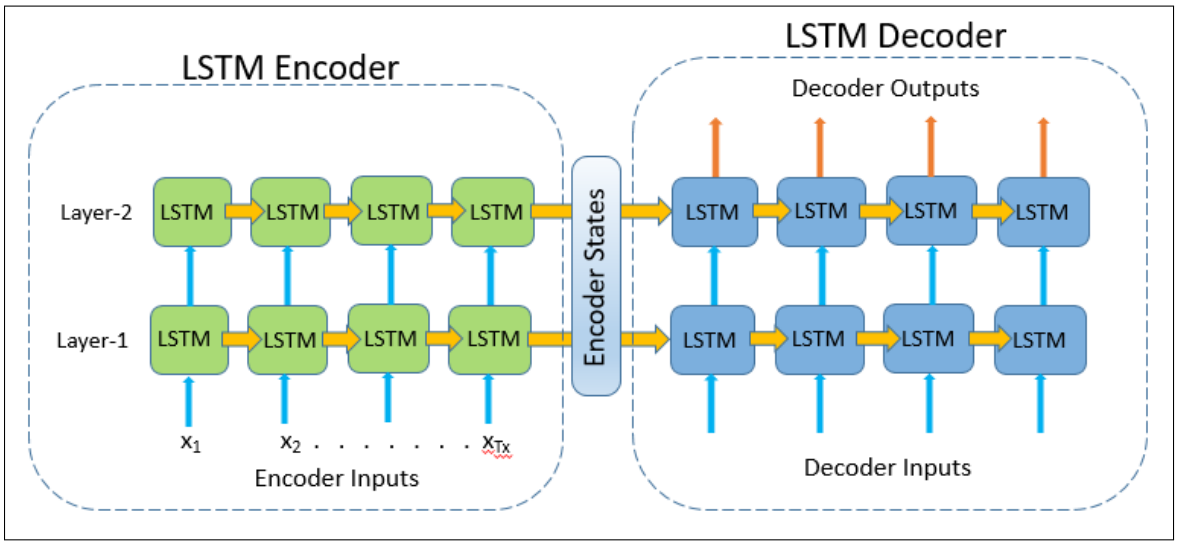


Figure 4.2: Schematic for a Deep Encoder-Decoder LSTM

4.1.2 Modular OpenRobots Simulation Engine (MORSE)

We made use of two different simulators for the purposes of this project. The first simulator which was the primary domain of all the experiments is the 'ModularOpen-Robots Simulation Engine: MORSE' [10]. As per the documentation available on the MORSE website, MORSE is an generic simulator for academic robotics. It focuses on realistic 3D simulation of small to large environments, indoor or outdoor, with one to tenths of autonomous robots. MORSE supports the use of various open-source middlewares such as ROS, YARP, Pocolibs, HLA, MOOS and Mavlink based on the architecture of the intended project. It also presents the option of being controlled entirely from the command-line.

Users of the simulator create Python scripts to generate simulations. The simulations so defined by means of user-written scripts are then rendered using the Blender Game Engine. MORSE contains a variety of standard sensors of the likes of laser scanners, GPS sensors, Odometry sensors and many visual sensors such as the RGB

camera and the Depth Cameras in its component library . The datastream that is output from these sensors can be processed by means of another Python script. The components also contain actuators such as the speed controllers, waypoint controllers, and joint controllers in addition to robotics bases such as quadrotors, the ATRV, PR2 and other generic 4-wheel vehicle models. The user can choose from multiple degrees of realism of the simulation they want to render based on the function the simulation serves.

For instance, vision applications that would make use of data from the camera sensors need to have accurate visual sensors whereas the motion controllers might be a secondary priority, enabling the choice of an easy to use Waypoint Controller. For the purposes of the experiments we conducted in MORSE, we made use of a couple of mobile robots, specifically the ATRV robots bases which played the roles of the Pursuer and the Evader respectively. Each of these bases is occupied with a Waypoint controller and a Laser sensor. The Pursuer bot also has attached to it an RGB camera sensor, the images from which are collected and processed by means of a callback function written inside a Python Script. The controllers are also actuated from within the same script, based on the datastream coming off the sensors attached to both the mobile robots.

4.1.3 Blender Game Engine

Blender [8] is a cross-platform, free, and open source 3D creation suite that works with Linux, Windows and Macintosh operating systems. It makes use of OpenGL for a consistent experience in 3D rendering tasks. As described above, MORSE supplies the models and animations for the robots bases, sensors and actuators that have already been built using Blender Engine. The default as well as the user-written python scripts are then deployed to the game engine using the Blender API to be rendered

as a simulation. The API also provides support for interaction with the simulation enabling communication with the sensors and actuators specified by MORSE scripts.

4.1.4 CARLA (Car Learning to Act)

Besides using the MORSE simulator as the primary domain for testing the prediction model, we also want to determine the extent of its generalizability. Therefore, we designed an experiment inside the CARLA simulator [9] which confirms the adaptability of the prediction model across multiple domains. CARLA is an open-source urban driving simulator built for research involving autonomous driving. CARLA also provides digital assets such as urban layouts, vehicles and buildings. These assets together form an environment. CARLA supports experimenting within several such environments containing varied placement of the digital assets in each environment mimicking different urban driving scenarios. CARLA also provides an array of sensors and actuators analogous to those provided by the MORSE simulator (cameras, depth sensors, LIDARs, GPS etc), that can be mounted upon the vehicle models and communicated with using the CARLA API. The autonomous vehicles in deployment are referred to as 'actors'. Scripts written in python can be deployed to govern the actions and behavior of said actors. For the architectures that require it, CARLA also supports integration with the Robot Operating System (ROS) via a ROS-bridge they provide. The specifications for the suite of sensors available in CARLA and the environments including the weather conditions, traffic simulation and road behaviors are also flexible.

For our experiment we deploy an actor (autonomous vehicle) with an RGB camera sensor for the data generated by the image stream. This data is used by the prediction model (deployed via the API using a python script) to be tested and the accuracy of the output is measured to ensure its inter-domain compatibility. Since

the experiment in CARLA is designed to expressly test the compatibility of the prediction mode, it hasn't been set up in the form of a Pursuit-Evasion game. Thus, unlike the experimental setup inside the primary domain, i.e MORSE, the LIDAR sensors do not need to be mounted explicitly atop the actors since we rely on the default navigation stack supplied by CARLA for driving through the environment.

4.2 Formulating the Problem

The game of pursuit-evasion where we aim to test our hypothesis is set up as follows. The time steps $t = 0, 1, ..., k, ...$ are discrete individual steps, corresponding to the ticks of the simulator. The space itself is formulated as a continuous two-dimensional Euclidean plane in R^2. That is because it is a Cartesian coordinate system formed by the orthogonal intersection of two lines where each is a copy of the real line R and the intersection is referred to as the 'origin'. The obstacles in the environment are cuboids (boxes and walls) for the sake of a simpler implementation. These obstacles represented by $Z = \{\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, ...\}$ are placed at random positions for every new episode of the simulation. Thus the mathematical representation of these obstacles is defined as $\mathbf{z}^{(i)} = (x_z^{(i)}, y_z^{(i)}, l_z^{(i)})$. Here, $(x_z^{(i)}, y_z^{(i)})$ is the reference to the center of the i^{th} obstacle in the Euclidean plane. The dimensions of each of the obstacles (l, w, h of a cuboid) are denoted by $l_z^{(i)}$, $w_z^{(i)}$ and $h_z^{(i)}$, where i enumerates a particular obstacle in the set. This concluded the definition of the passive elements of the pursuit-evasion game. Next are the active components, i.e the Pursuer $\mathbf{x}_p^{(k)}$, the Evader $\mathbf{x}_e^{(k)}$ and the properties that define them. Both the pursuer and the evader are 4-wheeled vehicle models spawned inside the MORSE environment. Mathematically, they are defined as rigid bodies with the following properties; the position of an agent on the R^2 plane (considering it as a single-point entity) referring to its x and y

coordinates, the direction the agent is facing toward, i.e, its orientation, and finally the agent's velocity. At any given time step k, the states of the pursuer and the states of the evader are defined as follows :

$$\mathbf{x}_p^{(k)} = (x_p^{(k)}, y_p^{(k)}, \theta_p^{(k)}, v_p^{(k)}), \quad \mathbf{x}_e^{(k)} = (x_e^{(k)}, y_e^{(k)}, \theta_e^{(k)}, v_e^{(k)}),$$

where $x^{(k)}$, $y^{(k)}$ are the x and y positions of the pursuer and the evader, θ and v are the orientations and speeds of the agents. We assume at a certain time step, the agent can only move forward at a specific speed v along its heading direction determined by θ and then change its current orientation by $\Delta\theta$, which is a simplified model of the vehicles driving on the road. Hence, $(v, \Delta\theta)$ are considered as the control input of an agent, which is generated by the navigation stack.

The perception model of the pursuer is defined as $g_p()$ for the environment whereas the $h_p()$ represents the perception model for the evader,

$$Z_p^{(k)} = g_p(Z, \mathbf{x}_p^{(k)}), \quad \hat{\mathbf{x}}_e^{(k)} = h_p(\mathbf{x}_e^{(k)}),$$

where, $Z_p^{(k)}$ is a set of observed obstacles, $h_p()$ represents a suite of sensors that can detect the evader and provide its current state $\hat{\mathbf{x}}_e^{(k)}$. The evader on the other hand, [has an environment perception model as below,

$$Z_e^{(k)} = g_e(Z, \mathbf{x}_e^{(k)}),$$

where $g_e()$ represents a sensor on the evader such as a LIDAR that can detect the cubical obstacles. It's counterpart $g_p()$ provides the pursuer with the same functionality. $Z_e^{(k)}$ is the set of detected obstacles (*i.e.*, $Z_e^{(k)} \subseteq Z$) by the evader. The evader also has the following motion model,

$$\mathbf{x}_e^{(k+1)} = f_e(\mathbf{x}_e^{(k)}, Z_e^{(k)}, \mathbf{u}_e),$$

where $f_e()$ represents its navigation stack and motion control module and $\mathbf{u}_e = (x_t, y_t)$ is the destination of the evader. Considering this particular implementation of the

pursuit-evasion game that contains a random-walking evader, the motion of the evader is governed only by the random goal locations sent to it and the obstacles along the path it takes to get to its destination. This implies that the evader moves independent of the pursuer's actions as depicted in the equations above. The perception and navigation systems incorporated in the agents is described in greater detail in chapter 4.3.

Other than the modules meant for perception and obstacle avoidance, since we base our approach after predatory beings that predict the future locations of their prey, we need to define a prediction module associated with the pursuer. This helps the pursuer make estimates about the future locations of the evader using an offline learned model as explained in detail in the next chapter. The observed locations of the evader using the perception module on the pursuer are stored and sent to this prediction module in order to make these sequence-to-sequence predictions. motion predictor that can estimate the evader's future states for m time steps, based the observed evader's states in the past n time steps, as follows,

$$P^{(k)} = h_{pp}(Q^{(k)}),$$

where $P^{(k)} = \{\hat{\mathbf{x}}^{(k+1)}, \hat{\mathbf{x}}^{(k+2)}, ..., \hat{\mathbf{x}}^{(k+m)}\}$ are the predicted states of the evader in the future, and $Q^{(k)} = \{\hat{\mathbf{x}}^{(k)}, \hat{\mathbf{x}}^{(k-1)}, ..., \hat{\mathbf{x}}^{(k-n)}\}$ is a set of states of the evader observed in the past. This prediction model abstracted as $h_{pp}()$, is an Encoder-Decoder LSTM, elaborated on in detail in chapter 4.5.

We also consider for comparison another prediction model that makes use of Kalman Filtering when provided with the kinematics of the evader. A trivial baseline case, where the pursuer does make use of any such prediction module, *i.e.*, $P^{(k)} = \emptyset$ is also evaluated. Next, we define the motion models of both our agents. We formulate the problem such that the pursuer has no knowledge of the evader's motion model.

The assumption here is that the experiments take place with an evader that has the same motion model as the evader on which the offline prediction module was trained. With the goal set as the predicted location of the evader, the pursuer moves with the intent to capture it. The motion model of this pursuer can be modelled as follows.

$$\mathbf{x}_p^{(k+1)} = f_p(\mathbf{x}_p^{(k)}, Z_p^{(k)}, \mathbf{u}_p^{(k)}), \quad \mathbf{u}_p^{(k)} = f_{pp}(\hat{\mathbf{x}}_e^{(k)}, P^{(k)}),$$

where $f_p()$ represents its navigation stack and the motion control module, similar to $f_e()$; $\mathbf{u}_p^{(k)} = (\hat{x}_t^{(k)}, \hat{y}_t^{(k)})$ is the pursuit policy represented as a target destination, and $f_{pp}()$ is the pursuit policy generator, which is explained further in chapter 4.6.What this policy generator $f_{pp}()$ essentially does, is that it determines the desired control input to be sent to the agent, by considering the presence of the predicted sequence of states of the evader obtained from the prediction module, or its absence thereof. In the meanwhile the navigation and motion control module $f_p()$ executes the generated policy to adjust the motion of the pursuer so as to capture the evader while avoiding obstacles.

The metaphorical 'capture' of the evader can be formalized as an event at a certain time step k as

$$||\mathbf{s}_p^{(k)} - \mathbf{s}_e^{(k)}|| < r,$$

, where $\mathbf{s}_p^{(k)} = (x_p^{(k)}, y_p^{(k)})$ and $\mathbf{s}_e^{(k)} = (x_e^{(k)}, y_e^{(k)})$ are the positions of the pursuer and the evader respectively, and r is a distance threshold typically set to a small value. For example this threshold is set to $r = 3$ meters in our experiment and can be varied based on the dimensions of the agents in play.

The criteria to compare the performances of two pursuers that employ different pursuit policies is that the pursuer with the lower capture time is better. The implicit assumption here is that both these pursuers have the same start states for a given episode. In this particular episode, the evader also spawns at the same initial location

and is supplied with the same sequence of goal locations from start to finish. Thus, finding a good prediction model $h_{pp}()$ and an effective policy generator $f_{pp}()$ is the main focus of this research issue.

4.3 Perception and Navigation

The agents involved in the symbolic game of cat and mouse need to be able to navigate unmapped environments that potentially change rapidly due to the unpredictable nature of the chase. The navigation module, therefore, has to have the capacity to to guide them to their goal location with precision. With pre-made maps of the environment out of the question, we need a navigation system that can take as input, a location relative to the agent's current location as input. Therefore as a precursor to initiating a navigation routine, an agent first needs to self-localize. In the real world, this can be done by making use of the Global Position System (GPS). Though officially committed to a higher tolerance, the actual accuracy of the localization using GPS signals is up to 0.715 meters, with a 95% probability. In terms of using the simulation, we use the 'Waypoint' navigation system provided by MORSE. This works by sending an agent the three-dimensional coordinates (z usually being equal to 0 for a flat plane) of the target's location in the world reference frame. This is achieved by first determining the agent's own location by pinging the simulation for it. Then the target's location relative to the agent (i.e from the agent's frame of reference), is added to its own absolute location to obtain the x, y and z coordinates of the target. This location is then sent out to the navigation stack. The job of the navigation stack is to first orient the agent towards this target location by turning the desired amount of degrees. Then a forward velocity is applied in the resulting direction. The navigation routine monitors the agent's own location to check if it

has reached the goal location. The routine terminates once this has happened. Calls made to the navigation stack with new goal locations when one routine is active, change the routine's destination and the previous goal is abandoned. We also need to account for the obstacles an agent might come across during the journey to its destination. This is where the LIDAR modules come in. Once the LIDAR detects an obstacle, an evasion routine is executed. This temporarily suspends the navigation routine and then resumes it once the hurdle has been cleared safely. This comprises the navigational capacity of an agent.

The agents also need to be able to "perceive" their surroundings (the pursuer in our case) to be aware of the opponent. The Perception Module, hence consists of a depth-sensing RGB camera. This module can be emulated inside the MORSE simulator by making use of the 'Semantic Camera' it provides. By definition, the semantic camera can determine the relative locations of predetermined targets, once they appear in a frame of the feed being captured by it. This bundles in two functions, the first being Object Detection where the target object (the adversarial agent in this case) is detected in a frame. The second function included in the semantic camera is that of the localization of the target. As explained in greater detail in the next section, this process would be similar to using an object-detection algorithm on the RGB frame, in tandem with the observed distance to the target obtained using the Depth frame, from a device similar to a Kinect in real-life conditions. The reason behind proposing this architecture is to allow the agents to function even in the absence of pre-mapped environments. Although navigation without the knowledge of the map poses a greater challenge for the development of the agents and pursuit algorithm, it also provides a much higher degree of flexibility.

4.4 Estimating the Evader's Current States

As defined in section III.A that describes the formulation of the problem, it is the function $h_p()$ that obtains the current states of the evader i.e $\hat{\mathbf{x}}_e^{(k)}$. The process of obtaining the location of the opponent (the evader) is subdivided into two separate steps. The first step is to detect the evader using the camera on board the pursuer. Once this detection is made, the next step is to then determine the location of the evader in terms of the world coordinates. We go about executing these steps as follows.

We make use of an RGB-D sensor such as a Microsoft Kinect or an Asus Xtion. These sensors generate two streams of frames. One stream is made of the RGB frames captured. Each of these frames has 3 channels corresponding to the Red, Green and Blue components, similar to any other normal camera. The other stream contains Depth frames. These are two-dimensional (1-Channel) frames, where the value of a pixel defines the distance to the object being observed in the corresponding RGB frame. Coming to the first step, we obtain an RGB image from the camera on the pursuer. Then we obtain a 2D bounding box around the evader in the image by subjecting the RGB frame to an object-detection algorithm such as YOLO [19] for example. YOLO is an object detection neural network popular for producing robust results and having a low computational cost on execution. This ensures that the algorithm can operate in real-time and can keep up with the frame rate of the camera. Once the bounding box is obtained, we get the coordinates to the center of said bounding box. Then we calculate the angle α between positions of the evader and the pursuer.

Besides calculating this angle, we also use the depth camera to obtain the distance d from the agents hosting the camera (the pursuer) to the target (the evader). Finally

making use of some simple trigonometry, we derive the evader's current location in the map.

$$\hat{x}_e^{(k)} = x_p^{(k)} + d * cos(\alpha + \theta_p^{(k)}) \quad \hat{y}_e^{(k)} = y_p^{(k)} + d * sin(\alpha + \theta_p^{(k)}).$$

Obtaining the location of the target hence provides us with the means necessary for the implementation of a pursuit policy. This policy shall dictate the utilization of the location so obtained, in both a direct and an indirect manner. In the implementation we propose, the estimated evader states are further processed acquire an estimate of the evader's possible future location (*i.e.*, $h_{pp}()$) and generate a pursuit policy (*i.e.*, $f_{pp}()$) based off of it.

4.5 Predicting the Evader's Future States

This part of the pursuit strategy lies at the nucleus of the solution to the pursuit-evasion game we formulate. The essence of this step is to obtain a tactical advantage over the evader. This advantage helps us mitigate the uncertainty in the evader's behavior and also helps the pursuer quickly close down on the lead, in terms of the distance to the evader. The task of the pursuer hence, also includes reasoning about the motion of the evader in order to be able to anticipate the places on the map the evader might eventually be, given a set of its current states.

Considering the motion-model of the evader $f_e()$ to be similar to what we routinely see in four-wheeled vehicles in real life provides us with the liberty to make certain assumptions regarding the evader's kinematics. We can therefore resort to using either model-predictive controllers or probabilistic state-estimation techniques such as Particle Filters or Kalman filters to predict the future position of the evader inside the environment. Since we work with the assumptions as mentioned before, we can

infer that our pursuer does not know the intricacies of the navigation stack represented by $f_e()$. This can mean an increase in the uncertainty of the predictions being made about the evader. Another factor that can contribute to a rise in the uncertainty of these estimates, is an increasing number of future steps that we aim to predict. We can reduce this uncertainty caused by not knowing the internals of $f_e()$ by observing the effects of it, using a collection of the evader's past locations. We can therefore summarize our overall approach to making the predictions about the evader as follows. The strategy here is to learn a representation of the actual model of $f_e()$ by using the samples of trajectories of the evader in an offline manner. This means that we create a different environment which is used only for collecting the training data and not during the actual experiments. This use of this structurally similar environment helps prevent over-fitting our learned model during the training phase. This is analogous to splitting available data into separate training and test samples. Inside of the training environment we set up the evader, in the same way as it is initiated during the experimental phase, i.e as a random-walking evader. The trajectories of this evader are recorded and then trained upon to learn the target representation of $f_e()$. This process is referred to as offline training. This trained model is stored and then loaded back on to the evader during the testing phase. It is thus able to use previously unseen data to make predictions on the go. We therefore call this phase online prediction.

The overall process is formulated as a Time Series Analysis problem. Since we use the locations of the evader observed by the pursuer in the past to predict the future locations, this is essentially a problem of predicting a sequence of values, when provided with another. More specifically this is a what is known as a Sequence to Sequence encoder. Here, at time k given the past observations $Q^{(k)} = \{\mathbf{x}^{(k)}, \mathbf{x}^{(k-1)}, ..., \mathbf{x}^{(k-n)}\}$, we predict the future time series $P^{(k)} = \{\hat{\mathbf{x}}^{(k+1)}, \hat{\mathbf{x}}^{(k+2)}, ..., \hat{\mathbf{x}}^{(k+m)}\}$. Several methods [23, 18][1][22] have been proposed to use deep learning techniques for time-series

forecasting and trajectory prediction. Here, we adopt an Long Short-Term Memory network based encoder-decoder model that is trained to reduce RMSE error $\mathbf{L}$, between the target trajectory and the predicted trajectory, where N is the number of samples, $\Delta\hat{\mathbf{x}}$ is the difference in positions of adjacent states of predicted values and ground truth $\Delta\mathbf{x}$ for a given window.

$$L = \frac{1}{N}\sum_{i=1}^{N}(\Delta\hat{\mathbf{x}} - \Delta\mathbf{x})^2 \tag{4.1}$$

Following is a detailed description of how the above process works inside the MORSE simulator. We set up an environment that has a randomized placement of obstacles using Blender, which is then loaded as a map with the initialization script. Inside this environment, we spawn a 4-wheeled agent. We attach a 'Waypoint' module to this agent that provides it with navigation capabilities and a LIDAR module that aids with obstacle avoidance. Random goals are sent to it, and the trajectory it takes to reach each of those locations is recorded and saved as a '.csv' (comma-separated values) file. This is done by appending a 'pose sensor' to the evader. The pose sensor returns the location (world coordinates) of the agent it is attached to, at fixed intervals of time. This frequency can be adjusted and is usually set to the 'tick' of the simulator. Other information, such as the orientation of the agent at the corresponding tick can also be derived using an IMU (Inertial Measurement Unit) and recorded.

As described in chapter 2, the goal here is to emulate the functioning of the dorsal anterior cingulate cortex of the brains of predators. We want to be able to create and store a representation of the future location of the prey (here the evader) and make use of the knowledge of its observed motion model to our advantage. We believe the

encoder-decoder LSTM fulfills this role in the system model we propose, in that it can abstract the function that takes in input similar to the one available to dACC neurons and produces an analogous output. We hereby have at our disposal, the data collected from a random-walking evader that was set up in a separate environment. We have also decided upon a prediction strategy, which is to think of the prediction strategy as a Time Series problem and use a Sequence to Sequence Encoder-Decoder model. This will let us provide the last n time-steps taken by the evader as input and obtain the next n steps it might possibly take, as output. We also decided on using an LSTM as this Encoder-Decoder model for its high accuracy and a close to real-time implementation. The LSTM is given a stack of recorded locations of the evader $Q^{(k)}$, to generate the prediction $P^{(k)}$, then use the actual trajectory of the evader $\{\mathbf{x}^{(k+1)}, \mathbf{x}^{(k+2)}, ..., \mathbf{x}^{(k+m)}\}$ as supervision to train the neural network over multiple epochs.

4.5.1 The Training Setup

All the different environments/maps, whether they belong to the same domain (MORSE) or are from different domains, have their own region boundaries. That is, the dimensions of these maps and the coordinates of objects inside them are subject to variation. The presumed point of origin (0,0) of the reference frame that generates world coordinates also changes from map to map. The map sizes and the coordinate systems differ even between the data collection, training and testing environments we use in MORSE. Moreover, since the placement of the obstacles in these environments is random, we cannot let our prediction model learn based on the exact coordinates of the evader in the environment. We therefore have to find a way to abstract the motion model of the evader to render it independent of these absolute coordinate values. Keeping this necessity in mind, we pass the differences between consecutive

locations of the evader instead of the locations themselves, as input to the model.

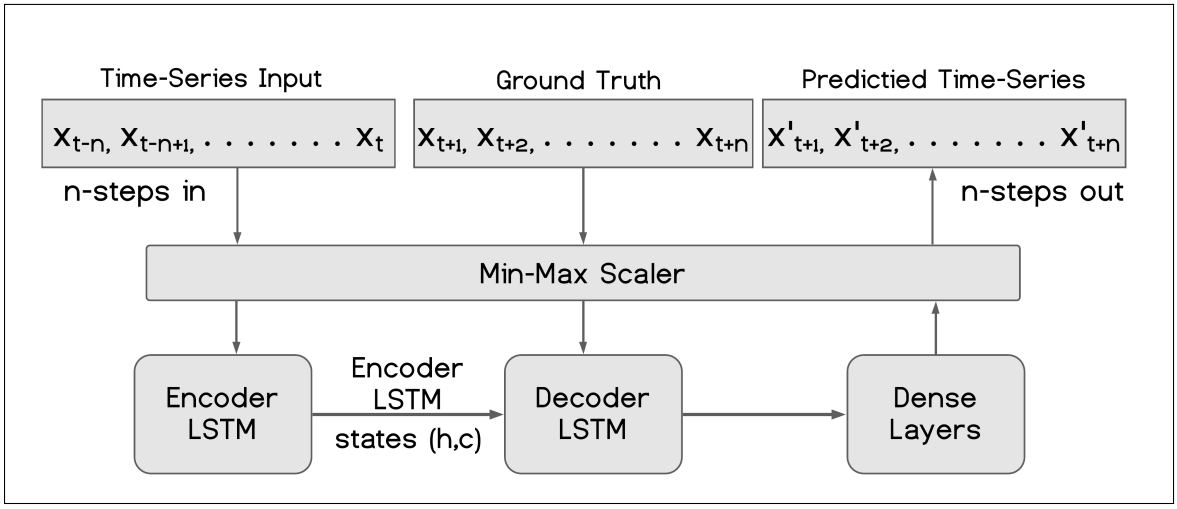


Figure 4.3: Training Schematic for the Encoder-Decoder LSTM

This is very intuitive as the differences between two consecutive locations capture the nuances of the agent's movement. For example, if the agent is slowing down, these differences decrease and are spaced further apart when the agent speeds up. Another important step to make this model usable across domains is to consider the range of the training data, against the data we are about to make predictions on. The maximum velocities the agents can travel at in an environment, depend on the simulator being used. For example, inside of our default environment MORSE, let's assume the speed limits are from a lower bound of $V_{M_{low}}$ to and upper bound of $V_{M_{high}}$. We want to implement this model in another domain (CARLA) for example, where the agents can move at speeds ranging from $V_{C_{low}}$ to $V_{C_{high}}$. The differences in consecutive locations of the evader therefore depends on the different speed ranges. Thus before we provide an One-Hot encoded input to the LSTM, we scale it to the range of data from the domain where it was trained on. This is done using a "Min-Max Scaler" as shown in Figure 4.3 . This ensures that the inferences about the motion model of the evader drawn during the training of the LSTM are preserved by scaling the test data during the test.

4.6 Policy Generation and Execution

In this chapter we discuss each of the three pursuit policies implemented in this project. As defined in chapter III.A, $f_{pp}()$ is our policy generation function. With regards to each generated policy, we define what decisions the pursuing agent takes during the course of the game.

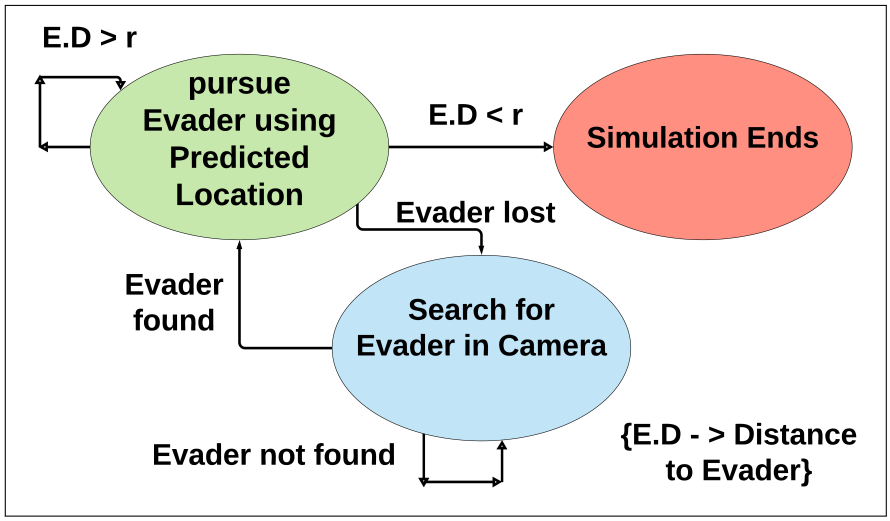


Figure 4.4: The Finite State Machine of the Pursuer.

4.6.1 Camera_Only

This model is very straightforward. When the evader is visible to the camera, a goal location is generated based on its estimated location on the map and sent to the pursuer's navigation stack. No locations are sent if the evader cannot be detected in the pursuer's camera.

4.6.2 Kalman Filter

This model generates the goal $\mathbf{u}_p^{(k)}$ as the position of the furthest predicted state $\hat{\mathbf{x}}^{(k+m)}$ in $P^{(k)}$, and $P^{(k)}$ is obtained using a Kalman filter with a constant velocity model of the evader. Whenever the evader is visible, the filter updates its belief based on the location estimated by the camera.

4.6.3 LSTM

Similar to the Kalman filter predictor, this model generates goal $\mathbf{u}_p^{(k)}$ as the position of the furthest predicted state $\hat{\mathbf{x}}^{(k+m)}$ in $P^{(k)}$, but $P^{(k)}$ is obtained using the LSTM motion predictor. When the evader is not visible in the camera we keep sending goal locations available in the prediction stack generated by the LSTM, until all these predictions are used.

There is always the possibility of the evader moving out of the pursuer's sight. Since this is not a search problem and simply a pursuit scenario, we consider the evader lost in such a case where it manages to escape the pursuer's field-of-vision. Once such a situation is detected, the pursuer waits for a stipulated amount of time after which the episode is terminated. If however, during this buffer the evader reappears in the pursuer's camera, the pursuit of the evader resumes and the termination of the episode is cancelled. This protocol is what the finite state machine diagram depicts in the state 'Search for the Evader in Camera' in Fig. 4.4. Other alternatives such as rotating in place until the evader is found again or invoking a separate search algorithm are not used so as to maintain the integrity of the initial pursuit formulation.

Chapter 5

EMPIRICAL RESULTS

5.1 Experiment Setup and Evaluation Protocol

As discussed at the start of chapter 4, we create different variants of a MORSE environment by randomizing the locations of obstacles in it and also randomly adding in walls in some of these environments. We also select an arbitrary empty spot (not occupied by obstacles or the other agent) inside the environment as a spawn point for the evader. It is then assigned to from the set of other empty spots in the environment, a goal location. This random sequence of goal locations is supplied to the evader for as long as the episode is active. The pursuer is spawned with the evader in its view. The policy generator described in chapter 4.6 enabled the pursuer to implement different variants (LSTM-based, Camera_Only or Kalman filter based) for the chase. We keep track of the coordinates between the Pursuer and Evader while these chases occur via a subscription to their locations. Finally we plot graphs comparing the distances between the Pursuer and the Evader for each path we test on. For the training of the LSTM, we collect the trajectory from a randomly moving evader with varying velocities inside a MORSE environment. We trained the encoder-decoder LSTM on this data with an 80:20 validation split, resulting in a mean validation RMSE of 0.019 meters.

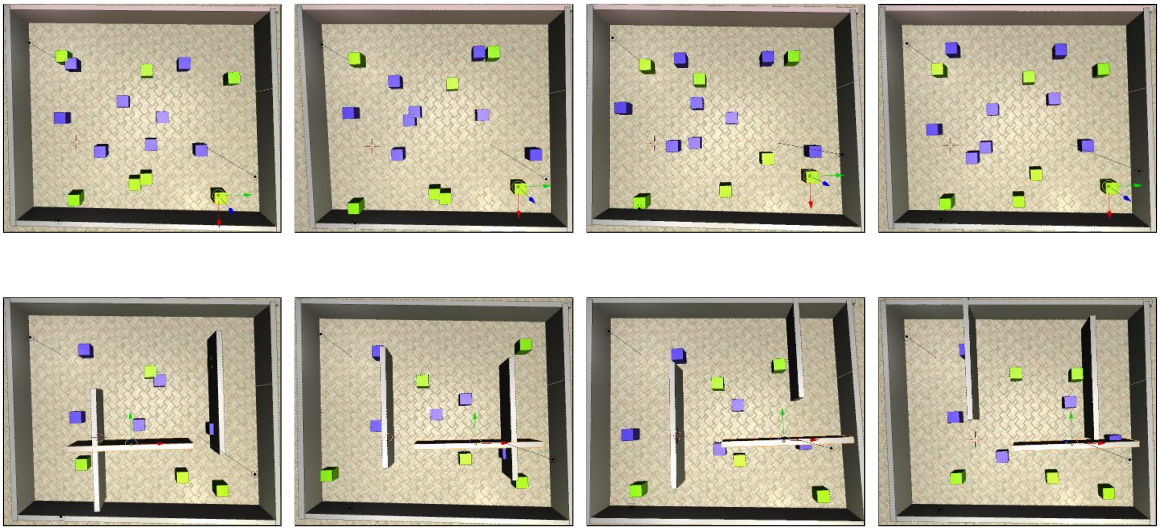


Figure 5.1: Environments with Box Obstacles (top row); Environments with Obstacles and Walls (bottom row).

5.2 Performance Evaluation

Performance analysis is a pivotal part of being able to solidify our hypothesis where we claim that 'a prediction based pursuer can capture the evader quicker, as compared to traditional models that make use of either a Kalman filter or those that simply use the location obtained from the pursuer's camera'. In order to obtain such a metric that we can evaluate for each of the three pursuit policies mentioned above, we conduct experiments per the setup described in chapter 5.1.

To summarize, we have three different pursuit policies, whereas the environments we create are of two different types. The first type is where the environment only has obstacles in the shape of boxes which we call 'Obstacles Only'. In the other type of environment we introduce a little more complexity by inserting walls in addition to the cuboidal boxes. We call this setting 'Obstacles + Walls'. We play out pursuit-evasion games between the two agents by varying the policy of the pursuer as well the type of environment. Each instance of this game is termed as an 'episode'. For every

episode corresponding to a certain pursuit policy, the positions of the agents and the obstacles in the MORSE environment is randomized. The environments and spawns are identical for a corresponding episode across all the pursuer and environment types to allow comparison between them. Data such as whether the evader was captured, the number of time steps taken to capture the evader, specifics of the environment used in each simulation (obstacle locations) and a continual representation of the relative distance of the robots. The results from these experiments are represented in Table 5.1.

Another thing to note here is that since each episode has a different starting distance between the pursuer and evader, the steps taken to capture an evader that is further away will be greater than the steps taken to capture an evader that is closer to the pursuer. We therefore also need to normalize the number of steps taken in an episode relative to the minimum number of steps required by the pursuer to capture a static evader at the same starting distance. We denote $G(i)$ - the steps taken by the pursuer to capture a static evader, $A(i)$ - the actual steps taken by the pursuer to capture the evader (moving) and $N(i)$ - the normalized steps taken by a pursuer to capture an evader in episode i. $N(i) = S_i * \frac{A(i)}{G(i)}$ where S_i is a binary variable that denotes if the pursuer captures the evader (1-captured, 0-not captured) in the episode i. Therefore, the mean normalized step count can be calculated as

$$C = \sum_{i=1}^{n} S_i * \frac{A(i)}{G(i)},$$

where n is the total number of test episodes. We report this count (and its variance) for all episodes in the results in Table 5.1. The lower the value of the normalized step count the faster the pursuing agent is at capturing the evader.

Environment	Mean Normalized Step (C)		
	Camera_Only	**Kalman Filter**	**LSTM**
	mean_steps, % success	**mean_steps, % success**	**mean_steps, % success**
Obstacles Only	0.363 ± 0.12, 95%	0.569 ± 0.21, 56%	0.266 ± 0.04, 89.5%
Obstacles + Walls	0.243 ± 0.05, 93%	0.319 ± 0.08, 56.5%	0.199 ± 0.03, 88%

Table 5.1: Normalized mean number of time steps to capture and success rate of capture for different Pursuers

For each of the variants of the pursuit-evasion game setup, we tested for 200 episodes. In the *Obstacles_Only* type of environment, the time steps to capture were less in 144 instances when the LSTM model was enabled as opposed to the Camera_Only pursuit policy and were less in 147 simulation instances when compared to the Kalman filter-based policy. For the environments with walls for obstacles in addition to just the cuboidal boxes, capture was quicker with LSTM than the camera_only and kalman filter approaches in 139 and 157 simulations respectively. Hence, the mean steps taken to capture using the LSTM model was significantly less compared to the other models.

Observable in table 5.1, is the capture rate for the LSTM-based pursuer, which is slightly lower than that of the camera_only model. We assign higher importance to lower mean steps taken rather than the success rate since the problem we model this solution for, represents time-critical situations. A pursuit scenario is an event of great uncertainty and high stakes, as there is a high probability of damage to life and property while the chase is active. This probability can be reduced by aiming to limit the time period for which the pursuit scenario lasts. This implies that a method that results in a quicker capture and puts an end to the chase as soon as possible is preferable. Slower capture of the evader also endangers pursuit by elevating the chances of the evader escaping. Hence the priority is to be able to terminate the

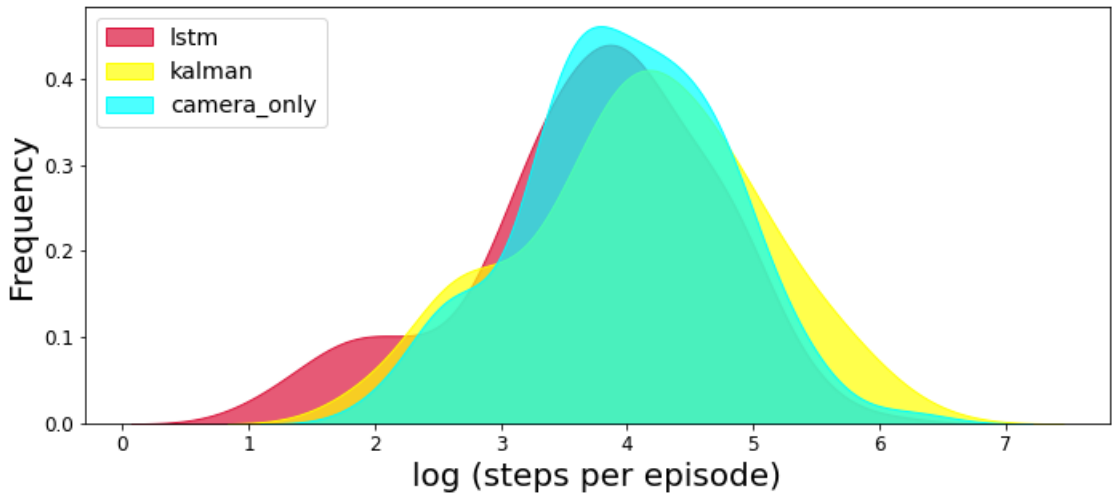


Figure 5.2: Analysis of mean steps to capture for different pursuers in the Obstacles + Walls environment

episode as early as possible with a capture. In addition to this, it is our hypothesis that the percentage of episodes that result in a successful capture of the evader shall even out as we test all three pursuers for a much larger number of episodes.

The Kalman Filter works by sacrificing its current view of the evader for a tactical advantage. This is a good strategy for evaders moving more or less in a straight line, as attested to by the results of our experiment, where in a few episodes the Kalman-based evader captures quicker than either of its counterparts but falls behind overall. This accounts the higher mean steps and a lower success rate for the Kalman-Filter model compared to its counterparts.

We also conducted a Student's T-test to determine whether the comparative results obtained for the different pursuit policies from our experiments were statistically significant. A T-test is performed to compare the means of two different groups. For our experiment, we compare the mean time steps for capture taken by the LSTM-enabled pursuer to its counterparts. In a T-test there is a null-hypothesis and an

alternate hypothesis which contradicts the former. The null hypothesis here is that the mean of time steps taken to capture using the LSTM and the other model we compare it to, are the same. Our aim is to be able to refute this null hypothesis by proving that the difference in means is not merely a coincidence. As an output of the T-test we compute the 't-value' and 'p-value' using the number of steps taken to capture in each episode, per pursuer. A p-value of less than 0.05 indicates a greater than 95% confidence to reject null hypothesis.

Environment	**Student's T Test**	
	kalman filter vs LSTM	**camera only vs LSTM**
	(p-value, t-value)	**(p-value, t-value)**
Obstacles Only	4.9×10^{-9} , 6.2	0.006 , 2.74
Obstacles + Walls	4.6×10^{-6}, 4.71	0.043 , 2.03

Table 5.2: Student's t-test comparing LSTM to other pursuers [preferred $p<0.05$ and $t>1$]

On the other hand, an absolute t-value of greater than 1 means the difference between the data sets is significant enough to reject the null hypothesis. The results from the Student's T-test based on the mean steps to capture for the different pursuers are depicted in Table 5.2. The t-value received from the test comparing the camera only model with the LSTM approach was around 2.74 with the obstacle_only setup and around 2.03 with the walls added, indicating that there is a significant difference in the results received when comparing.

The p-values were 0.006 and 0.043 (rounding off to three significant digits) for the same two cases, indicating a very low probability that the results received occurred by chance. This solidifies our hypothesis that the LSTM model-based capture task

takes less time steps to complete compared to all other models and also establishes that the results from the experiment are not coincidental.

5.3 Domain Transfer Task

Another aim of this experiment is to see how readily transferable our trained LSTM is, to different domains. We already made provisions that render this implementation of the prediction model independent of the exact map coordinates of the opponent. Using the minmax scaler on the input to the prediction model also makes it possible for our agent to track and predict of evader's of speeds varying across different ranges. To make sure this hypothesized domain transfer is also empirically consistent, we test the same prediction model in a domain other than the home domain of MORSE.

In this alternate simulator CARLA, we spawn a model vehicle with a Pose Sensor attached to it. This pose sensor polls the location of the vehicle and returns it via a callback. This enables the collection of the agent's trajectory via storing the data retrieved on every callback that gets triggered at a fixed frequency.. The locations from the trajectory are then buffered and sent to the predictive model (pre-trained in MORSE) for estimating future locations of the evader. We draw these predicted locations as well as the actual path the evader took besides each other, to visually validate the accuracy of the predictions. This attests to the fact that our method is applicable to dynamic scenarios that are analogous to a rapid change in domains. This entropy in the environment where the pursuit-evasion game takes place, often rules out conventional solutions like mapping the environment in advance or using reinforcement learning to train the agents. The transferability shown by our model makes it a viable solution even in the face of challenging environmental circumstances.

Inside the CARLA Simulator a vehicle is spawned at a random location on the city map. It is also given an arbitrary location on the map as the destination. CARLA comes with its own navigation stack which bundles navigation tasks with obstacle avoidance. There is hence no necessity to attach a LIDAR to the vehicle and the prediction module consisting of an RGBD camera shall suffice. The location from the vehicle is sent to the prediction model and the error between the predicted values and the known ground truth is recorded.In the experiments we conducted in CARLA we tested the prediction model for over a 100 episodes. The mean RMSE over these 100 runs was 0.138 meters which is well within the acceptable limits considering the average length of a car is about 4.2 meters. This establishes the adaptability of the prediction model to domains other than MORSE.

Chapter 6

FUTURE WORK

This project has helped us take a significant step in the direction of a future containing robust Autonomous Vehicles that can operate without restrictions in all kinds of challenging scenarios. There definitely are ways in which we can enhance our current system.

Our current approach to navigation in MORSE is simply to match the orientation to the goal location and then start traveling along that direction in a straight line until we reach the destination. This makes it a difficult task to maneuver across bigger obstacles (large obstacle-size to evader-size ratio). Since there is no map of the environment available, there can be no predetermined path planned around the obstacle. The obstacle avoidance routine that we implement in this paper is an active routine, which thus has the potential to last for long periods of time, essentially terminating the chase. A possible solution to this would be to enhance the pursuer's perception with the ability to create temporary 3D maps of the area directly in front of it. We can then use this short-term model to plan a path around the bigger obstacles. This could be accomplished using an RGB-D sensor that keeps the setup low-cost.

Other live-mapping techniques could be used to triangulate and store the positions of obstacles in the environment as the pursuer moves around in it. This shall help reduce bad predictions from the model by ignoring those that coincide with stored locations of obstacles.

Another improvement that can be made is to incorporate a way to estimate the evader's current orientation, as an input to the prediction of its future location. Since the orientation presents the evader's general heading, using it in addition to the

current coordinates of the evader can greatly improve the accuracy of the predictions. This automatically reduces the probability of predictions that are not in the same direction as the evader's estimated orientation and can pave way for lengthier output prediction sequences for comparatively shorter inputs. This means that the time that the evader needs to be observed in order to generate predictions reduces significantly and the time step for which it can reliably predict the evader's location is further into the future.

Another concern we aim to address is the motion policy of the evader. So far, we have considered different pursuer models and compared their performance against an evader which has always been of the 'random-walking' kind. Evaluating the pursuer against an intelligent evader, i.e one that actively evades pursuit, shall be a critical step in further development. What we can say about the current model, however, is that it treats all the evaders in the same way. In defense of a random-walking evader, it could be claimed that the trajectory generated by a random motion policy can sometimes coincide with the output of an active evasion policy. One way to improve the performance for our current pursuer that is agnostic to the evader policy, can be to incorporate the orientation of the evader into the prediction schema as discussed above. This is because the evader's orientation is a clear indicator of the nature of the evasive maneuver it is about to execute. We, therefore, believe that this enhancement to the pursuer's perception arsenal will account for a more intelligent policy from the evader. We also intend to further this work by considering the effect of using prediction techniques in scenarios where there are more than one pursuers/evaders involved. Coordination between the agents on either of the teams based on the future estimation on the other team is a greatly interesting research avenue. It opens up many possibilities such as one where a team is able to 'set traps' for another in advance, based on their predicted locations, all in real-time.

Chapter 7

CONCLUSION

As a culmination of this work, we build a visibility-based Pursuer which can function without requiring a map of the environment. We successfully validate our hypothesis that this pursuer does a better job of capturing the evader using an encoder-decoder LSTM, as opposed to pre-existing estimations techniques such as linear dynamic estimation with Kalman filter, by estimating its future location . We also show that we can adopt the prediction model across domains without the need to retrain using any information from the new domain. This formulation of a pursuer-evader game under an autonomous vehicle setting and its validation environments facilitate future avenues of research.

REFERENCES

[1] Alahi, A., K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", (2016).

[2] Baker, B., I. Kanitscheider, T. M. Markov, Y. Wu, G. Powell, B. McGrew and I. Mordatch, "Emergent tool use from multi-agent autocurricula", CoRR **abs/1909.07528**, URL `http://arxiv.org/abs/1909.07528` (2019).

[3] Bhattacharya, S. and S. Hutchinson, "On the existence of nash equilibrium for a two-player pursuit-evasion game with visibility constraints", International Journal of Robotics Research **29**, 7, 831–839, URL `http://journals.sagepub.com/doi/10.1177/0278364909354628` (2010).

[4] Bilgin, A. T. and E. Kadioglu-Urtis, "An approach to multi-agent pursuit evasion games using reinforcement learning", in "Proceedings of the 17th International Conference on Advanced Robotics, ICAR 2015", pp. 164–169 (IEEE, 2015), URL `http://ieeexplore.ieee.org/document/7251450/`.

[5] Catania, K. C., "Tentacled snakes turn c-starts to their advantage and predict future prey behavior", Proceedings of the National Academy of Sciences **106**, 27, 11183–11187, URL `https://doi.org/10.1073/pnas.0905183106` (2009).

[6] Chen, B., S. Song, H. Lipson and C. Vondrick, "Visual hide and seek", arXiv preprint arXiv:1910.07882 (2019).

[7] Cho, K., B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation", CoRR **abs/1406.1078**, URL `http://arxiv.org/abs/1406.1078` (2014).

[8] Community, B. O., *Blender - a 3D modelling and rendering package*, Blender Foundation, Stichting Blender Foundation, Amsterdam, URL `http://www.blender.org` (2018).

[9] Dosovitskiy, A., G. Ros, F. Codevilla, A. M. López and V. Koltun, "CARLA: an open urban driving simulator", CoRR **abs/1711.03938**, URL `http://arxiv.org/abs/1711.03938` (2017).

[10] Echeverria, G., N. Lassabe, A. Degroote and S. Lemaignan, "Modular OpenRobots Simulation Engine: Morse", in "Proceedings of the IEEE ICRA", (2011).

[11] Emil, K. R., "A new approach to linear filtering and prediction problems", Transactions of the ASME–Journal of Basic Engineering **82**, Series D, 35–45 (1960).

[12] Esposito, W. R., *Hamilton–Jacobi–Bellman equationHamilton–Jacobi–Bellman Equation*, pp. 1473–1476 (Springer US, Boston, MA, 2009), URL `https://doi.org/10.1007/978-0-387-74759-0_258`.

[13] Guibas, L. J., J. claude Latombe, S. M. Lavalle, D. Lin and R. Motwani, "A visibility-based pursuit-evasion problem", International Journal of Computational Geometry and Applications **9**, 471–494 (1996).

[14] Hespanha, J. P., M. Prandini and S. Sastry, "Probabilistic pursuit-evasion games: A one-step nash approach", in "Proceedings of the IEEE Conference on Decision and Control", vol. 3, pp. 2272–2277 (IEEE, 2000), URL `http://www.ece.ucsb.edu/{~}hespanha/published/one-nash.pdfhttps://ieeexplore.ieee.org/abstract/document/914136/`.

[15] Kolling, A. and S. Carpin, "Solving Pursuit-evasion Problems with Graph-Clear : an Overview", pp. 2–6 (Citeseer, 2010), URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.655.3779{&}rep=rep1{&}type=pdf`.

[16] Mischiati, M., H.-T. Lin, P. Herold, E. Imler, R. Olberg and A. Leonardo, "Internal models direct dragonfly interception steering", Nature **517**, 7534, 333–338, URL `https://doi.org/10.1038/nature14045` (2014).

[17] Patsko, V. and V. Turova, *Homicidal Chauffeur Game: History and Modern Studies*, pp. 227–251 (2011).

[18] Rangapuram, S. S., M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang and T. Januschowski, "Deep state space models for time series forecasting", in "Advances in Neural Information Processing Systems 31", edited by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi and R. Garnett, pp. 7785–7794 (Curran Associates, Inc., 2018), URL `http://papers.nips.cc/paper/8004-deep-state-space-models-for-time-series-forecasting.pdf`.

[19] Redmon, J. and A. Farhadi, "Yolov3: An incremental improvement", CoRR **abs/1804.02767**, URL `http://arxiv.org/abs/1804.02767` (2018).

[20] Ruiz, U., R. Murrieta-Cid and J. Marroquin, "Time-optimal motion strategies for capturing an omnidirectional evader using a differential drive robot", IEEE Transactions on Robotics **29**, 1180–1196 (2013).

[21] Russo, J. F. and S. A. Sheth, "Deep brain stimulation of the dorsal anterior cingulate cortex for the treatment of chronic neuropathic pain", Neurosurgical Focus FOC **38**, 6, E11, URL `https://thejns.org/focus/view/journals/neurosurg-focus/38/6/article-pE11.xml` (01 Jun. 2015).

[22] Sutskever, I., O. Vinyals and Q. V. Le, "Sequence to sequence learning with neural networks", CoRR **abs/1409.3215**, URL `http://arxiv.org/abs/1409.3215` (2014).

[23] Tadayon, M. and Y. Iwashita, "Comprehensive analysis of time series forecasting using neural networks", (2020).

[24] Vidal, R., O. Shakernia, H. J. Kim, D. H. Shim and S. Sastry, "Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation", IEEE Transactions on Robotics and Automation **18**, 5, 662–669, URL http://ieeexplore.ieee.org/document/1067989/ (2002).

[25] Wang, Y., L. Dong and C. Sun, "Cooperative control for multi-player pursuit-evasion games with reinforcement learning", Neurocomputing **412**, 101–114, URL https://doi.org/10.1016/j.neucom.2020.06.031 (2020).

[26] Weintraub, I. E., M. Pachter and E. Garcia, "An introduction to pursuit-evasion differential games", (2020).

[27] Wikipedia contributors, "Rufus isaacs (game theorist) — Wikipedia, the free encyclopedia", https://en.wikipedia.org/w/index.php?title=Rufus_Isaacs_(game_theorist)&oldid=991073500, [Online; accessed 18-March-2021] (2020).

[28] Wikipedia contributors, "Torrence parsons — Wikipedia, the free encyclopedia", https://en.wikipedia.org/w/index.php?title=Torrence_Parsons&oldid=1000093212, [Online; accessed 18-March-2021] (2021).

[29] Xu, L., B. Hu, Z. Guan, X. Cheng, T. Li and J. Xiao, "Multi-agent Deep Reinforcement Learning for Pursuit-Evasion Game Scalability", in "Lecture Notes in Electrical Engineering", vol. 592, pp. 658–669 (2020), URL https://doi.org/10.1007/978-981-32-9682-4{_}69.

[30] Yoo, S. B. M., J. C. Tu, S. T. Piantadosi and B. Y. Hayden, "The neural basis of predictive pursuit", Nature Neuroscience **23**, 2, 252–259, URL https://doi.org/10.1038/s41593-019-0561-6 (2020).

[31] Zhang, Z., Y. Sung, L. Zhou, J. M. Smereka, J. Lee and P. Tokekar, "A minimax tree based approach for minimizing detectability and maximizing visibility", CoRR **abs/1807.09437**, URL http://arxiv.org/abs/1807.09437 (2018).

[32] Zhu, J., W. Zou and Z. Zhu, "Learning Evasion Strategy in Pursuit-Evasion by Deep Q-network", Proceedings - International Conference on Pattern Recognition **2018-Augus**, 67–72 (2018).