Prediction is very difficult, especially about the future.

– Neils Bohr (1885 - 1962).

University of Alberta

Predicting Opponent Locations in First-Person Shooter Video Games

by

Stephen Michael Hladky

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Fall 2009
Edmonton, Alberta

Examining Committee

Vadim Bulitko, Computing Science

Michael Bowling, Computing Science

Marcia Spetch, Psychology

Abstract

Commercial video game developers constantly strive to create intelligent humanoid characters that are controlled by computers. To ensure computer opponents are challenging to human players, these characters are often allowed to cheat. Although they appear skillful at playing video games, cheating characters may not behave in a human-like manner and can contribute to a lack of player enjoyment if caught. This work investigates the problem of predicting opponent positions in the video game Counter-Strike: Source without cheating. Prediction models are machine-learned from records of past matches and are informed only by game information available to a human player. Results show that the best models estimate opponent positions with similar or better accuracy than human experts. Moreover, the mistakes these models make are closer to human predictions than actual opponent locations perturbed by a corresponding amount of Gaussian noise.

Acknowledgements

I would like to thank the following people for their support and encouragement during my studies. This thesis could not have been completed without their efforts.

- To my supervisor **Vadim Bulitko** for giving me the opportunity to investigate artificial intelligence in video games. Having become excited about the possibility of working with Counter-Strike gameplay data at a graduate student orientation, little did I know that it would become the focus of my research for the next few years. I am grateful for the guidance, encouragement, and the utmost patience he offered me throughout my academic career.

- To the members of the Counter-Strike Artificial Intelligence (CSAI) research group for their insights and suggestions for improving the quality of my research. Specifically to **Michael Bowling** for providing a third-party perspective on my work and to **Jeffery Grajkowski** for the code used to calculate lines-of-sight and parsing file structure in Counter-Strike maps.

- To my roommate and colleague **David Thue** for offering me a place to stay, partaking in the many philosophical discussions about our research, and putting up with my messy lifestyle.

- To the members of the Intelligent Reasoning, Critiquing, and Learning (IRCL) research group, specifically **Alejandro Isaza**, **Greg Lee**, **Shanny Lu**, and **Chris Rayner**, for their support and companionship during each of our journeys in the pursuit of knowledge.

- To **Michael and Kimberly Bombak**, **Barrett Rodych**, **Matthew Hillier**, and all the members of the Edmonton Kendo and Naginata Club (EKNC) for ensuring that I remained sane by giving me a dose of the outside world every once in a while.

- To my parents, **Joseph and Christine Hladky**, and the rest of my family for all the phone calls filled with encouragement, assurances, and love.

- Finally, to the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Informatics Circle of Research Excellence (iCORE) for their generous funding of my research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Skillfulness and Believability

Creating a computational system that demonstrates human-like intelligence is a fundamental goal for Artificial Intelligence (AI) research. A traditional approach to this problem has been to design AI for small, well-defined domains that humans consider to be challenging, for example board games. Several AI systems have been developed that surpass the best human players at such games: Deep Blue for chess [50], Chinook for checkers [56], TD-Gammon for backgammon [59], and Logistello for othello [11] to name a few.

Unfortunately, board games alone are an insufficient measure of human intelligence. Although they test the proficiency of a player's planning and decision-making abilities, such games are not concerned with other human activities considered "intelligent" such as verbal communication, facial expressions, and physical movement. An alternative approach to AI focuses on achieving human-like intelligence by designing agents that exhibit realistic human behaviours. This approach is evident in virtual reality (VR) simulations where computer-controlled humanoid agents interact within a three-dimensional virtual environment. Many successful commercial VR simulations provide humans with immersive training in fields such as military combat experience [38] and sports practice [40]. However, assessing the performance of an AI agent's behaviour in VR applications is a non-trivial task.

With the increase in processing power and ubiquity of personal computers in recent years, a new medium for developing human-like intelligence has emerged: computer video games. Video games are an ideal testbed for human-like intelligence because they reconcile the divergent aims of the aforementioned approaches. Specifically, AI agents should be *skillful* at the game by making strategically and tactically sound decisions. As well, AI agents should be *believable* by behaving in a similar manner to other human players. This work proposes that AI agents who embody both of these attributes enhance the realism of the game and contribute to overall human player enjoyment.

The main contribution of this thesis is the design and implementation of computational models for predicting opponent locations in First-Person Shooter (FPS) video games. The models are

tested in real-world scenarios using Counter-Strike: Source, a popular online tactical shooter video game. Model performance is evaluated using two measures based on the notions of skillfulness and believability mentioned above. Finally, the models are shown to be comparable to human experts at predicting opponent locations and fit for use in commercial video games.

In this chapter, Section 1.2 introduces video game AI and identifies properties that make AI development challenging. Section 1.3 discusses how "cheating" techniques have been used to address these properties. The problem of predicting opponent locations in FPS games is motivated in Sections 1.4 and 1.5. Finally, the contributions of this thesis are listed in Section 1.6.

1.2 AI in Video Games

Although there are many different types of video games, ranging from casual play to realistic simulations, this work focuses on games that utilize three-dimensional virtual environments. In these games, human players typically interact within the game environment through an *avatar*, an in-game entity whose actions can be manipulated by use of a hardware interface (e.g., mouse, keyboard, gamepad). For example, an avatar might take the form of a sports car in racing games, a fantasy creature in role-playing games, or a professional athlete in sports games. In each of these cases, the player typically perceives the environment from the perspective of the avatar and may only take actions that are afforded by the avatar. In essence, the avatar can be considered as the player's virtual manifestation within the game world.

One purpose for AI in computer video games is to provide computer-controlled avatars called *bots*. Bots may be used in a variety of ways to produce enjoyable gameplay. For example, bots can help enhance the realistic aspects of the game world, such as simulating life-like behaviour for crowds of people in urban-themed environments [7]. As well, bots may aid story-telling by providing rich, compelling characters with whom human players can engage in conversation [60]. Finally, bots may be used as substitutes for human players, serving as allies in team-oriented games or as opponents in adversarial games. This latter purpose is of particular interest because these bots must provide an adequate challenge for human players (i.e., demonstrate skill). Also, because these bots emulate human behaviour, their actions must appear as if a human performed them (i.e., demonstrate believability). This work takes a step towards creating bots that appear both skillful and believeable.

1.2.1 Video Game Challenges

Video games are a challenging domain for bots because of several issues that must be addressed to produce skillful and believable behaviour.

- **Real-time scenarios.** Video games with 3D virtual environments often operate in real-time and as such bots must perform their current action while deliberating on the next one. This property renders a number of standard algorithms in planning and pathfinding inapplicable as

they cannot guarantee a fixed amount of deliberation time per action.

- **Large game states and action spaces.** Classical games such as Chess and Checkers have finite state spaces; that is, every possible state of the game can be enumerated. There are also a finite number of actions a player can take on his turn to transition among the game states. These two properties provide a foundation for *search*, a special class of AI techniques used to decide on appropriate actions to take during games. Oftentimes video games do not share these properties and thus pose significant challenges. Specifically, the state space can be continuous and the number of available actions in each state may also be vast and/or continuous. Thus, an intelligent bot should be able to recognize important features of the state space and use them to inform its decisions in the action space.

- **Imperfect information.** Recall that human players experience the game world from the perspective of their avatar. An avatar's field-of-view is often limited to a small area of the game environment and thus a correspondingly small portion of the game state is made known to the controlling player at any given time. Similarly, bots must be able to make decisions using only the information provided by their avatar.

- **Teamwork.** Cooperative multiplayer games require teams of players to work towards a common goal. Although players have the freedom to ignore their teammates, it is generally agreed that coordinating player actions allows a team to engage in better strategies that are not available otherwise, thus contributing to a higher probability of winning. A bot that aims to coordinate with other players must have a prior knowledge of effective strategies, be able to determine which strategy its teammates are currently executing, and then assume an appropriate role that contributes to the execution of the chosen strategy.

Due to these challenges, it would seem difficult to design a bot that can consistently outperform expert human players in these games. Nonetheless, there are several commercial games that offer levels of AI difficulty that are nearly impossible to defeat. How is this possible? One simple reason is that developers allow bots to cheat.

1.3 Cheating in Video Games

Cheating is a common activity in all forms of games. In physical sports, athletes have been known to take performance-enhancing substances (e.g., steroids) to become faster and/or stronger than their opponents. If one considers the economy to be a game, cheating could be viewed as a stock trader violating financial laws (e.g., reporting incorrect income, laundering money). In all cases, the temptation to "break the rules" of the game is spurred by a potential benefit for the offending player. In the former example the athlete wagers celebrity fame while in the latter the trader increases his

personal wealth. With regards to video games, entertainment value is a deciding factor in the commercial success of a product and thus developers use cheating techniques in their bots as a means to guarantee player satisfaction. In adversarial games, the process of confronting and overcoming a challenging opponent is considered to be a source for player enjoyment. However, the difficulties posed by the points in Section 1.2.1 coupled with the reality of intense production schedules make development of challenging AI problematic. By providing bots with unfair advantages (discussed in Section 1.3.1), developers can ensure a high degree of bot skill with the intention of delivering satisfying gameplay for human players.

Unfortunately, there are certain undesirable consequences that come as a result from cheating. If exposed, doping athletes may be expected to relinquish their awards and their fame might suffer accordingly; a thief can be convicted of stealing resulting in fines and/or imprisonment. Video games are also subject to the drawbacks of cheating. From an academic perspective, designing bots that cheat is not considered progress towards practical artificial intelligence. Because cheating bots rely on the game environment to favour their actions, they may not be able to perform successfully in other unbiased domains. And, while some argue that video games do not require "full-blown" AI [29] but rather only sufficient intelligence to maintain fun, if a bot is caught (or even perceived as) cheating, then humans may be left feeling slighted or frustrated, ultimately leading to a loss of player enjoyment.

Note that games may not provide both humans and bots with the same type of avatars. Consider a fantasy role-playing game (RPG) where a human player controls a knight about to do battle in a cave with a giant dragon controlled by a bot. If the dragon were to be endowed with superior attack and defense abilities (e.g., deadly fiery breath and thick scales that can withstand the knight's sword), any direct combat with the dragon would certainly result in the knight's demise. While the knight does not share the dragon's abilities, it is difficult to accuse the bot of cheating because any perceived unfairness could be attributed to poor game design.[1] However, in games where bots are used as substitutes for humans, expert players have a solid understanding of what is possible with their avatar. Bots that cheat in these games are at a greater risk of being caught because any bot behavior that a human cannot emulate would be identified as suspicious.

1.3.1 Examples of Cheating

Despite the consequences of cheating, many video games still provide their bots with unfair advantages in several ways. Because bots reside within the game itself, they can be easily endowed with perfect accuracy and reactive control over their avatars. In sports games, an AI-controlled team can always produce a pass or score a goal if necessary in situations where it would be nearly impossible for humans players to do so. Likewise, soldier bots in combat games can ensure that they never miss when firing their weapons. The bots are not subject to the human delay between perceiving the

[1] Typically, developers would design the game in a way such that players could defeat the dragon through an amazing feat (e.g., collapsing the cave on the dragon) thus ensuring fair gameplay.

game world and executing actions nor do they need to provide their actions through crude manual interfaces.

In the above examples, the act of cheating involves the enhancement of a bot's abilities. This type of cheating can be detected by *directly* observing a bot's actions. However, cheating may occur in subtler forms too, such as accessing and manipulating "unknowable" parts of the game state. For instance, an important challenge faced by all players in real-time strategy (RTS) games is the collection of resources for building structures and units. This challenge can be mitigated for bots by artificially inflating the amount of resources they acquire. A player's total resources are typically hidden from opponents and thus this type of cheating must be detected by *indirect* means (e.g., reasoning about the number of possible units one can construct given the average resources available and the time elapsed during the game).

Ultimately, if a bot cheats by enhancing its abilities or acting on hidden information, it risks taking actions that would not (or could not) be made by human players in the same situation. For example, if a cheating poker bot is given information about the cards held in the hands of its opponents, the bot would always choose the correct action of folding when the opponents have higher hands. However, without this hidden information, the alternative actions of betting and calling may also offer statistically similar chances of winning. This work investigates the hypothesis that bots can perform both skillfully and believably without cheating.

1.4 First-Person Shooters

One game genre that manifests all of the challenges presented in Section 1.2.1 (and thus makes non-cheating AI very difficult to design) is First-Person Shooters (FPS). Popularized in 1992 by the release of the game Wolfenstein 3D [42], the FPS genre has grown such that games are often hosted at international tournaments. For example, the FPS games Halo 3 and Counter-Strike are listed as official events by the World Cyber Games organization [43]. This professional level of competition is indicative of a strong fan base that enjoys playing FPS games.

FPS games are characterized by two key features which, unsurprisingly, compose the genre title. First, each player interacts with the game from a first-person perspective; that is, they perceive the environment through the "eyes" of their avatar (see Figure 1.1). Avatars in FPS games typically take on a humanoid form, affording the controlling player abilities such as running, jumping, climbing, and other basic human movements. Second, the primary form of entertainment provided by FPS games is firearm-based combat. Combat action takes place on virtual battlefields where avatars assume the role of infantry soldiers. Each avatar is outfitted with (or can acquire) a varied selection of weapons and equipment for the purpose of eliminating opponents from the game.

Figure 1.1: *Counter-Strike: Source*, a first-person shooter video game developed by Valve Corporation [48]. Note how the player views the game world from the perspective of his avatar.

1.5 Predicting Opponent Locations

Developing FPS bots is a hot topic in both the AI research community and video game industry. Ideally, the behaviour of an FPS bot should be indistinguishable from that of an expert human player and thus a bot should should emulate the complex human faculties involved when playing FPS games. One faculty of interest is the estimation of current opponent positions – an important task in games with partially observable game states [15, 12]. Knowing the location of opponents in FPS games is very useful for several reasons: it can be used to determine the best strategy to play, serve as a guide to direct a player's field-of-view, and ensure survivability as the chance of being caught unprepared is significantly reduced. From an academic perspective, predicting opponent locations is a challenging task because the complex trajectories of human opponents must be modeled in real-time over long periods without any knowledge of the opponents' true locations.

1.6 Thesis Contribution

The main contribution of this work is the design and implementation of computational models for predicting opponent locations. These models function as a stand-alone module, imparting a reasoning capability to bots with the aim of achieving skillful and believable behaviour. Specifically, hidden semi-Markov models and particle filters (introduced in Chapter 4) form the core of these modules with two key enhancements. First, the modules are machine-learned from records of past

matches, informing the models of the common direction and velocities of opponents (Section 3.3.3). Second, these modules are not allowed to cheat, being informed only by the same sensory game information that is available to a human player (Section 3.3.2).

The computational models are tested in actual recorded Counter-Strike: Source matches, presently the most played online tactical shooter video game. The performance of the models are evaluated by two measures, based on the notions of skillfulness and believability (Section 2.3). To compare these models with expert humans at the same prediction task, a user study is conducted in which experienced human players are asked to provide their own guesses of opponent positions. The results of the study and the performance of the models are presented in Chapter 6 followed by a discussion and directions for future work in Chapter 7.

Chapter 2

Problem Formulation

The goal of this work is to design *predictors*, computational modules used to predict the positions of opponents during FPS matches. This chapter presents a formal description of this prediction task, referred to as the Opponent Location Prediction (OLP) problem. First, Section 2.1 proposes three desirable properties that a good prediction representation should have. Next, the OLP problem is detailed and a formal definition of a *predictor* is provided in Section 2.2. The chapter concludes by presenting two performance measures based on the notions of skillfulness and believability which are used to evaluate predictors in Chapter 6.

2.1 Prediction Representation

Although there are several ways to represent a prediction, this work uses a single point on the real plane $\mathbb{R}^2$ to denote an opponent's position. This formulation is desirable because it satisfies the following three properties. First, predictions are simple to illustrate. It is common in many games to display event notifications on a radar or "minimap" in a corner of a human player's view screen (Figure 1.1 has one in the top-left corner). Map coordinates are easily represented on a minimap, making them an intuitive and practical means for conveying positional information in a real-time setting.

Second, predictions are easy for humans to specify. The user study (described in detail in Section 6.2) requires humans to perform the same task as a predictor and thus the specification process should be as effortless as possible, lest participants become frustrated and withdraw from the study. The data collected from the user study allows direct comparisons to be made between human and machine predictions.

Finally, predictions are comparable to other predictions. Formally, (P, D) forms a metric space where P is the set of all possible predictions (i.e., $\mathbb{R}^2$) and $D : P \text{ x } P \rightarrow \mathbb{R}$ is an arbitrary distance function. A metric space provides a convenient way to measure the relative quality of predictions. For example, one can assert that prediction p_1 is "better" than prediction p_2 according to a baseline p_3 if $D(p_1, p_3) < D(p_2, p_3)$.

2.2 Opponent Location Prediction

The Opponent Location Prediction (OLP) problem is characterized by two adversarial teams engaging each other in a closed three-dimensional game environment. One team is arbitrarily designated the *friendly* team and the other the *opposing* team. The purpose of a predictor is to emulate the human capacity to track members of the opposing team (referred to as opponents) while they move about the environment. At any time during a match, the predictor must be able to output a *prediction vector* $A = (a_1, a_2, ..., a_n), a_i \in \mathbb{R}^2$, a list of points estimating the current positions of all live opponents.

A predictor operates from the perspective of the friendly team and it may utilize any information known to a member of the friendly team to inform its predictions. Specifically, a predictor is provided with the following game data as input:

- n: the number of opponents currently alive. It is common for FPS games to allow everyone to view real-time statistics such as the scores, names, and alive/dead status of all players (see Figure 2.1);
- t: the current time elapsed since the start of the game;
- O: the set of regions that at least one friendly player can view by an unobstructed line-of-sight. While it may seem unreasonable to assume that a predictor should be able to know the exact areas of the map that all teammates can see, it is common for experienced human teams to use voice communication to keep members informed of current observations and combat action [18]. Although there exist models for managing the transmission of information between separate entities (e.g., sharing observations among players through information particles [24]), this work foregoes this complication, focusing on the prediction task instead;
- S: the positions of opponents that occupy a region in O. When any friendly player has an unobstructed line-of-sight to an opponent, the opponent is said to be *sighted* and the opponent's positional coordinates are made known to the predictor. All opponents are considered to be indistinguishable from each other;
- D: the positions of opponents that have been eliminated from the game.

This list of game variables defines the extent to which a predictor can be informed by the current game.[1] This formulation prohibits predictors from cheating; that is, a predictor cannot access additional game information nor can it affect the gameplay in any way.

[1] The current map and the team being predicted are assumed to be global knowledge.

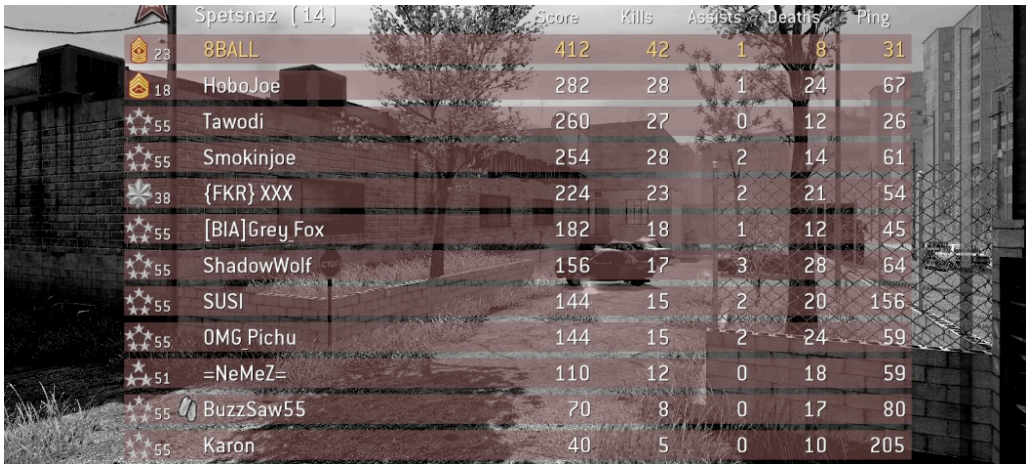


Figure 2.1: A screenshot taken from Gamespot.com [39] illustrating the scoreboard for the FPS game *Call of Duty 4* [1]. Note that the scores, names, and "dead" status of all players (denoted by an image of dog tags to the left of the name) are visible on this screen and can be accessed at any time during gameplay.

2.3 Performance Measures

The prediction representation discussed in Section 2.1 provides an objective approach for measuring the success of a predictor. To evaluate the quality of a prediction, the prediction point can be compared to a *reference* prediction point. The error incurred by a prediction with respect to a reference prediction is the in-game distance between the two points. Specifically, this distance is the shortest path traversable by an avatar that does not intersect walls or other impassable obstacles in the game environment.

If two or more opponents are present, then the error is defined as the average distance between a matching of points from a prediction vector to those of a reference vector (Figure 2.2). In all cases, the matching that minimizes the average distance is chosen. Thus, predictors with low errors can be said to make smaller mistakes and should be preferred over those with high errors.

Section 1.1 proposed that a bot should make skillful decisions while at the same time behave in a human-like fashion. Inspired by these desirable properties, predictors are evaluated according to two error measures: the *prediction accuracy error* (PAE) and the *human similarity error* (HSE). The former error denotes the "lack of skillfulness" for a predictor while the latter denotes the "lack of believability". The defining feature of each measure is its selection of reference predictions, which is discussed in the following sections.

2.3.1 Prediction Accuracy Error

To construct a reference vector for an error measure, one must consider the extreme case where a prediction vector should incur no error at all. With regards to skillfulness, an optimal predictor should

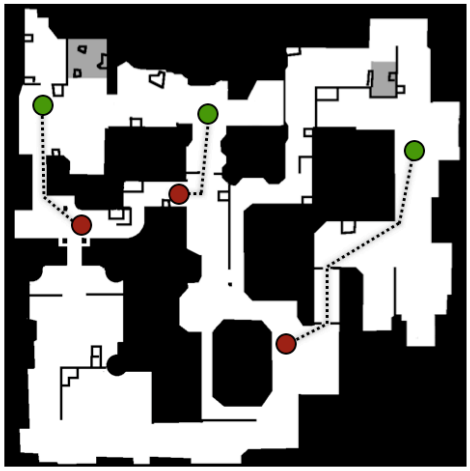


Figure 2.2: An example matching of a prediction vector (green circles) to a reference vector (red circles). Dotted lines indicate the shortest in-game path between matched pairs.

correctly estimate the exact location of every opponent at all times. Thus, the reference vector for the PAE measure would be the vector consisting of the true location of opponents. Definition 2.3.1 presents a formal description of an optimal predictor.

Definition 2.3.1. A predictor is *perfect* if every prediction scores a PAE of 0.

2.3.2 Human Similarity Error

Like the PAE metric, the HSE also requires the definition of a reference vector to be complete. However, this prompts the question "What are the most human-like predictions possible at any given point during a match?". Although there are several ways to quantify the "believability" of a bot's performance (discussed in detail in Section 3.4), the assumption is made that the most human-

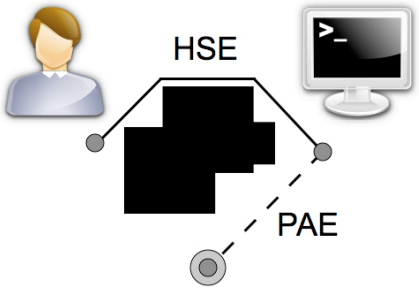


Figure 2.3: Relationship of error measures. The bottom point denotes the true position of an opponent while the other points denote the predictions of a human expert and a computer predictor. The dashed line represents the prediction accuracy error and the solid line represents the human similarity error.

like prediction is one made by a human expert. Therefore, a reference vector for the HSE metric would be a prediction vector provided by a human expert assuming the role of a predictor. These predictions are collected as part of the user study described in Section 6.2. Figure 2.3 shows an example of how both error measures relate to the true location of an opponent.

Chapter 3

Related Work

This chapter provides a review of research and development related to the design of a computational predictor. Section 3.1 begins the review with a discussion of *object tracking*, a general class of problems of which the OLP problem is a special case. Four features are identified here that characterize the OLP problem. Next, Section 3.2 analyzes commercial implementations of FPS bots with particular attention paid to any opponent prediction faculties they may offer. *Spatial-temporal* models are presented in Section 3.3 as potential solutions to the OLP problem. Motivation is provided for *perception reasoning* and *opponent modeling*, two concepts that can be incorporated into spatial-temporal models to produce informed predictions. Finally, existing methods for testing the skillfulness and believability of bot performance are reviewed in Section 3.4.

3.1 Object Tracking

Object tracking can be described as the problem of estimating the trajectory of an object as it moves around on an image plane [66]. Oftentimes, this task plays a role in many larger tracking problem domains, such as monitoring troop deployments for military surveillance [9], tracking hand movements for gesture recognition [65], and identifying moving obstacles for path planning in robotics [14, 5].[1] Each of these domains has different goals, assumptions, and constraints and it appears that no single solution is adequate to address object tracking in general. This section identifies four features that can be used to characterize object tracking problems. Although this selection of features is by no means comprehensive, it does provide a basis for comparing the OLP problem to other object tracking tasks.

Feature 1: Number of Targets

A *target* is an object to be tracked. Targets receives the attention of the tracking system (or *tracker*), be it a machine or human. As the number of targets increases, so too must a greater tracking effort be exerted by the tracker. While a computer's effort may be constrained by its processing

[1] See Yilmaz et al. for an extensive review of object tracking research [66].

power, humans have a limit to their tracking abilities. Pylyshyn and Storm claim that humans can successfully track in real-time up to five targets that appear identical to each other [51]. Because the user study discussed in Section 6.2 allows only five players per team as well as an indefinite amount of time for specifying predictions, the task of predicting opponent positions should not be beyond a participant's natural ability.

Feature 2: Environment Topology

Obstacles present on the image plane can also affect the difficulty of the tracking task. On planes with no obstacles, a target is free to move in any direction, which may prove challenging if its direction and velocity can change abruptly (see Feature 3). In contrast, FPS game environments are often constructed as a maze of hallways and connecting rooms. The direction of a target's trajectory is restricted in narrow corridors, which may benefit the tracker by allowing him to concentrate on targets in open areas.

Feature 3: Motion Complexity

Motion complexity refers to the policy governing a target's movement. Consider two targets: a billiard ball and floating grain of sand in a pond. The ball's trajectory is easy to predict: it should continue in a straight line after it has been hit by the cue, only changing direction when collision occurs with a buffer of the billiard table or another ball. Unlike the billiard ball, the sand grain constantly changes direction in the pond according to Brownian motion. However, predicting its motion is also easy as it can be achieved using a simple random walk model. Simple policies are often used in object tracking studies, such as a target that follows a circular path [63].

Goal-based movement is an example of a more complex motion policy. Bruce and Gordon track humans who move along an optimal path between two of 17 possible locations [10]. A good tracker must be able to discern the goal of a human given the current trajectory observed. The movement of players in FPS games are even more complicated given that any goal location they attempt to arrive at may be interrupted by combat, the announcement of a global event, or the time elapsed during the current match. Each of these interruptions may cause the player to change strategies and hence their direction and velocity. Thus, a tracker that deals with complex motion policies should be aware of important environment features and understand how these features can affect a target's movement.

Feature 4: Sighting Frequency

In object tracking studies, the limits of a human's tracking ability is tested by increasing the speed of target movement [63] and including additional moving objects called *distractors* for the purpose of disrupting the human's concentration [51, 2, 19]. In real-world scenarios, computer systems are able to successfully track objects with fast, complex motions using image recognition techniques (e.g., tracking balls in play for Foosball [67] and Robocup Soccer [30]). However, it is

common in all these situations that targets remain in plain sight. When a target's position is occluded from the tracker's view, the object tracking task becomes an object *prediction* task, similar to the OLP problem. The prediction task is made more difficult by increasing the time between sightings of the targets.

3.2 Commercial Approaches

To track opponents in FPS video games, several different techniques have been implemented in commercial products. This section investigates a few of these techniques and evaluates their ability to address the OLP problem.

Figure 3.1: Left: A "big daddy" approaches the player in *Bioshock* [35]. Right: The player views an open area containing multiple opponents in *Quake II* [41].

In the critically-acclaimed single-player FPS game *Bioshock* (Figure 3.1), the AI system simulates a community of bots that act autonomously [35]. Bots are designed to engage any character deemed hostile, be it the human player or another bot. With regards to OLP, a bot uses two vision cones to track each of its targets [13]. A *doubt* vision cone is centered on the last known location of the target and the bot looks in the direction of the cone when it decides to search for the opponent. In contrast, a *certainty* cone corresponds to the bot's current field-of-view. If the target remains in this cone for a certain length of time, the bot has "sighted" the target and can take appropriate action knowing the target's exact location.

Unfortunately, the cone model is limited in its player tracking ability. If the bot is unable to find any opponents in the doubt cone, the bot resumes its previous tasks, effectively "forgetting" that it sighted an opponent. In essence, the cone model does not account for the movement of opponents while they are out of view.

Laird addresses the above issue with his Quakebot, a bot designed to play the popular online FPS game *Quake II* (Figure 3.1) [26, 41]. The Quakebot is based on the Soar cognitive architecture, a production system designed to operate in real-world scenarios [28]. In Soar, a *reasoning context* is used to represented the current state of the world and system. Information in the context may include

beliefs about the current world state, goals waiting to be achieved, and additional information that has been deduced (or rather produced) from the application of production rules. This latter process is referred to as "elaborating the context". Coupled with the context is a *knowledge base* designed to store the production rules. A rule can be an action the system can take in the world (called an *operator*) or a method for generating subgoals with the aim of achieving a greater goal.

While the reasoning context and knowledge base form the core of Soar, several extensions have been implemented to add or enhance cognitive features. The Quakebot demonstrates the addition of *anticipation*, a subroutine used to predict the future actions of opponents [26]. When an opponent is sighted, the subroutine creates a temporary context in which the bot places itself in the opponent's position. Soar then operates as if it were the opponent, creating goals and taking actions in the game. The sequence of actions is recorded and returned to the bot after a specified time length, serving as a prediction of the human's future trajectory. This information is now placed in the reasoning context and can be elaborated with the hope of producing informed decisions.

Unlike Bioshock bots, the Quakebot does track opponents when they are not in sight. However, the opponent trajectories returned by the subroutine are only correct if the goals and decision-making processes of the opponents are accurately encoded as Soar operators. If this is indeed the case, then it would seem that the Quakebot should also behave similarly to human opponents. A panel of eight judges reviewed video clips of the Quakebot in action but were unable to draw any significant conclusions about the "humanness" displayed by the bot [27].

Wray et al. also use Soar as the decision system for their MOUTBot, an agent used to train United States Marines for military operations on urbanized terrain (MOUT) [64]. In contrast to the Quakebot, the MOUT bot maintains a *threat object* for each encountered opponent. A *record-threat* operator is executed whenever a relationship changes between the bot and one of its opponents. For example, when an opponent moves in or out of view of the bot, the appropriate threat object is updated with information such as where and under what conditions the sighting occurred. However, similar to the Bioshock bots, no proactive estimation of target positions are made while opponents are not in view.

3.3 Spatial-Temporal Models

One issue common to the implementations described in the previous section is that no analysis of the entire game environment is made. Very specific estimates (e.g., a sequence of coordinates) of an opponent's current and future positions are made and if it is discovered that the opponent does not occupy any of these locations, there is no alternative prediction that can be provided. In addition, these implementations do not update their models as time passes (with the exception of the Quakebot), accounting for the possibility of opponents moving while out of sight.

Spatial-temporal models are a means of representing data that has both location and time-dependent features. These models are well-suited to track players moving about the environment

and thus they form the basis of the predictors presented in Chapter 5. The following sections discuss variants of spatial-temporal models, noting their advantages and disadvantages when applied to the OLP problem.

3.3.1 Influence Maps

An *influence map* is a method for modeling the effect of a game feature on the environment over time. Typically, the environment is partitioned into a finite set of regions and a value is maintained for each region corresponding to the "influence" that the feature has on that region. These values can be updated to reflect the changes in the feature's influence as the game progresses.

Influence maps are frequently used in commercial FPS games. Figure 3.2 shows two examples from the games Halo 3 and Team Fortress 2, highlighting the areas of the map where players commonly eliminate others from the game (or are eliminated themselves). Colours are often used to illustrate the relative intensity of a feature, making it easy for players to analyze weakpoints in their own strategies and that of other players.

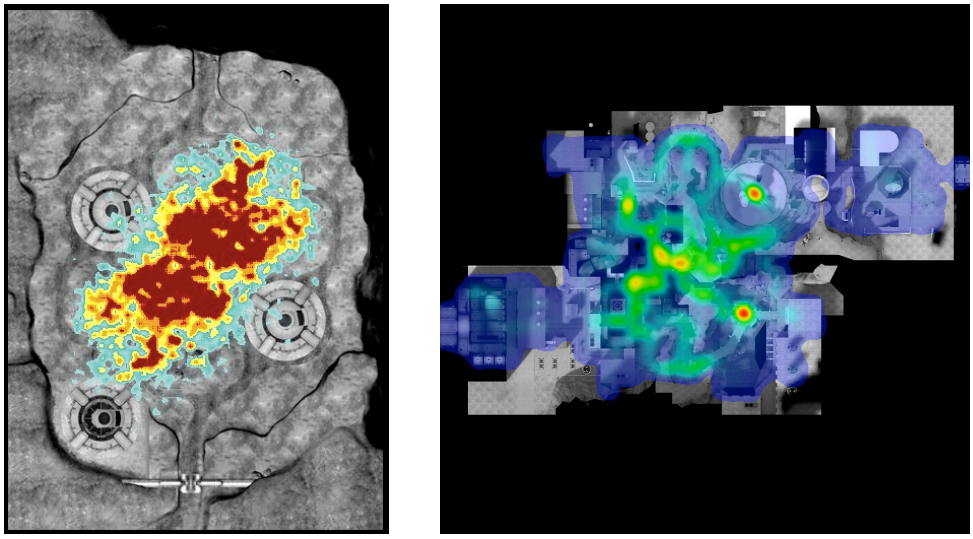


Figure 3.2: Left: An influence map from *Halo 3* showing areas from which a player is likely to eliminate another from the game [37]. Right: An influence map from *Team Fortress 2* showing areas where players are likely to be eliminated [47]. In both images red hues denote high probabilities while blue hues denote low probabilities.

Another feature that can be represented with influence maps is the notion of *threat*. For instance, the official Counter-Strike: Source bot avoids dangers areas when pathfinding through the game environment [8, 45]. Whenever combat is seen or heard by the bot, a threat value is increased in the appropriate map region (and subsequently decays over time at a fixed rate). These threat values are used to inflate the estimated cost for traversing the region; a large cost will ensure that an alternative path is found through areas with lesser amounts of threat.

Hoobler et al. created Lithium, a visualization framework for overlaying feature information in the FPS game Return to Castle Wolfenstein: Enemy Territory [20]. While the framework has the ability to render several game features depending on the user's preference, the most practical feature is *occupancy*. In this mode, regions are displayed in the team colour of nearby players. A blend of the team colours is used if two or more opposing players occupy the same or adjacent regions.

By themselves, influence maps are not designed to predict opponent locations. The occupancy feature is assumed to be known at all times for all locations in the environment. In the following sections, two techniques are presented that enhance influence maps when the occupancy feature cannot be fully known, making them viable solutions to the OLP problem.

3.3.2 Perception Reasoning

Doherty and O'Riordan suggest that a realistic bot should utilize the sensory information provided by its avatar to reason about the game environment [16]. Although other avatar sensory information such as audio or collisions can be considered, this work only focuses on visual observations. With regards to the OLP problem, a prediction should agree with all observations made about the game environment, both past and present. Specifically, confirming that an opponent is not in a given area is valuable information for informing predictions. For example, consider a room with a single entry point. If an opponent has been sighted moving into the room and the entrance to the room is under constant surveillance, a reasonable prediction would place the opponent somewhere within the room. In this way, even if a length of time passes without any opponent sightings, observations can still be used to constrain an influence map so that the opponent's position remains in the boundaries of the room. This analysis of observations, which will be referred to as *perception reasoning*, aids in answering the question "what are the *possible* locations that an opponent can occupy?".

Isla suggests the use of *occupancy maps* (an influence map modeling the occupancy feature) to predict opponent positions [22]. When a friendly player observes a region that does not contain any opponents, the probability corresponding to the region is set to zero and all other probabilities are normalized. The values of newly observed regions are updated in a similar manner as friendly players move throughout the map.

Isla evaluates occupancy maps as a means for guiding the search behaviour of bots [21]. In a game environment filled with obstacles, an AI-controlled dog is instructed to find a roaming sheep avatar controlled by a human. If the dog does not have a direct line-of-sight to the sheep, the dog moves to the region with the highest probability to continue its search. If the sheep is not sighted in this region, the appropriate probability is reduced to zero and dog heads towards the new region with the largest associated probability. This sequence is repeated until the sheep is found or the search task is discarded in favour of pursuing another goal. Isla claims that this "emergent" policy results in intelligent search behaviour however no experiments were run to support the statement. Nonetheless, this process for updating region probabilities is incorporated into the predictors presented in

Chapter 5.

3.3.3 Opponent Modeling

Having used perception reasoning to restrict an opponent's position to unobserved areas of the environment, a logical question to ask is "what are the *probable* locations that an opponent would occupy?". Ideally, predictions should be made along the opponent's planned trajectory. As such, any information about an opponent's preferred motion can be used to discount areas that are not frequented, further refining prediction accuracy.

In general terms, *opponent modeling* is the task of learning and encoding an opponent's behaviour for the purpose of informing decisions. Opponent modeling has been used to enhance AI in many different games. For example, research in Texas Hold'em poker has focused estimating which class of card-playing strategies an opponent is currently playing [4, 57]. As well, modeling a human player's preferences has been used to construct dynamic stories in role-playing games [61]. To address the OLP problem, the predictors presented in Chapter 5 focus on modeling *direction* and *velocity*, two important features for specifying player movement.

There are several ways to build models for opponents in FPS games. The simplest method is to assume the opponent's actions are *uniform*; that is, there is an equal probability of heading in any direction regardless of the opponent's current position. Isla makes this assumption in his description of occupancy maps [22]. Unfortunately, this method does not take advantage of actual game dynamics. For example, while it is theoretically possible for an opponent to stand in a corner for the duration of a game, one would instead expect him to enact strategies and tactics that increases his chance of winning the game. Moreover, the model is limited in that it can only track opponents moving at a single, constant velocity. An evaluation of uniform models is presented in Section 6.4.2.

Bererton uses particle filters (discussed in Section 4.4) as a means to predict the current locations of human opponents [6]. Specifically, particles serve as candidate positions that an opponent can occupy and account for opponent movement by spreading throughout the environment according to a Brownian (i.e., uniform) motion model. Bererton emphasizes that particle filters are an efficient method for tracking opponents, noting that 31 targets were tracked simultaneously during tests. However, no rigorous experiments were performed to confirm the accuracy of the motion model. Instead, performance benefits are supported only by anecdotal evidence.

Another way to construct an opponent model is to specify common game strategies and tactics by a set of hand-coded rules. Darken and Anderegg promote the construction of *simulacra*, motion policies that are designed to emulate a specific opponent behaviour [15]. For example, a "hider" simulacrum would have an agent take refuge at one of several locations known to be good hiding places. In contrast, a "hunter" simulacrum would instruct the agent to close the distance between its enemies once they have been sighted. However, representing complex behaviours requires both expert knowledge and significant effort on the part of the designer to specify precise rules.

Southey et al. use hidden semi-Markov models (discussed in Section 4.3.1) to track the movement of enemy units in a real-time strategy (RTS) game [58]. Specifically, a model is constructed for every pair of regions on the map and each model encodes the path that a unit would take when moving from one region to the other. The model that best fits the unit sightings and observations of the environment is used to predict the unit's trajectory. Experimental results show success in predicting unit trajectories, however this may be due to the significant number of sensors placed throughout the map (see Figure 3.3). As well, units were modeled as using an optimal path for traveling between goal regions. In contrast, predicting opponent positions in FPS games is more challenging because opponent sightings are infrequent. Moreover, FPS players are not guaranteed to follow the shortest path to goal locations. For example, in the event that players from different teams engage each other, their former trajectories end up disrupted by combat.

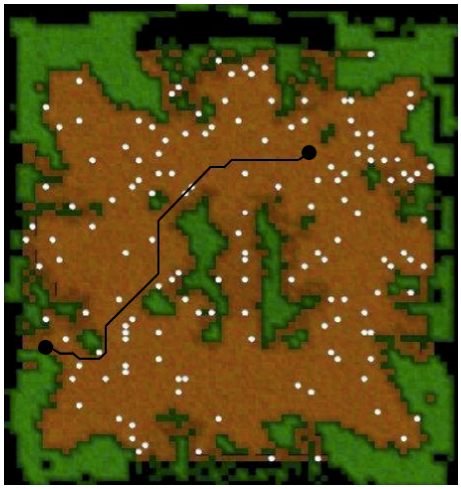


Figure 3.3: A sample Warcraft 3 map used in experiments by Southey et al. [36, 58]. The black line denotes an example path by an RTS unit. The white dots denote sensors which have an observation radius of 2.5 times their size.

In recent years, machine learning has received significant attention as a method for specifying opponent models. This method is desirable because the analysis of human behaviour can be done automatically and account for any idiosyncrasies in human movement that might otherwise be forgotten by experts. The predictors presented in this thesis are trained on gameplay data provided by humans as are the motion models described in the following works.

Bruce and Gordon also use particle filters to track the movement of human beings in an office environment [10]. Before tracking begins, a corpus of human trajectories is separated into clusters by hand. These clusters contains paths that are deemed similar to each other and are used to generate a set of goal locations where players are likely to originate from or head towards. The particle filter models the likelihood of a target following any particular cluster which can then be used to estimate the goal locations of the target. This learned model is shown to be superior at predicting human trajectories compared to a Brownian model. However, it should be noted that the experiment envi-

ronment is small and contains few obstacles. Large FPS environments can accommodate many goal locations and long trajectories, making the clustering task impractical for humans. Although Rayner presents an automated method for clustering player trajectories in Counter-Strike: Source [54], his work focuses solely on opening gameplay.

As mentioned in Section 3.1, intelligent robots must deal with moving objects when planning paths through their environments. Bennewitz et al. train a robot to recognize humans from a camera input and avoid collisions by using a hidden Markov model trained on example human trajectories [5]. Experiments are carried out in an office-like environment where humans move about 8 different goal locations (Figure 3.4). The probability of a human occupying a region is used as a coefficient when computing the cost of traversing the region during path planning. As such, the robot will move such that its path avoids the estimated location of humans.

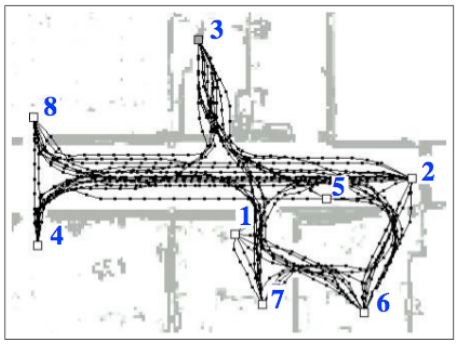


Figure 3.4: An office environment used in experiments by Bennewitz et al. [5]. Goal locations are labeled by numbers and example human trajectories are denoted by solid dotted lines.

Despite the fact that this research claims to utilize human generated trajectory data, the nature of these trajectories are simpler than that which is characteristic of FPS games. Specifically, humans always moved at a constant velocity, no paths involved backtracking, and long trajectories were segmented by goal locations where humans could rest for arbitrary lengths of time. The gameplay data used in this work to train predictors is not subjected to any of these conditions which preserves the complexity of human movement but increases the difficulty to capture this information in a opponent model.

3.4 Skillfulness and Believability Testing

Section 2.3 presented two metrics for evaluating the performance of a prediction. While these metrics are designed for the OLP problem, determining the skillfulness and believability of a bot's actions is a much more complicated matter. With regards to skillfulness, games provide an inherent measure of success, namely the final game result (win or loss). However, this measure requires the results of several games to assert that a bot's skill is statistically significant. Instead, other game features may provide a more specific breakdown of a bot's skill such as the number of opponents

eliminated during a game or an overall score that awards points based on completed goals and objectives. Overall, games provide several objective ways to measure skillfulness, unlike believability which is evaluated subjectively.

The classic method for determining how believable a machine is at imitating humans is the Turing test [62]. In a generalized formulation of the test, an interrogator is allowed to interact with two subjects through an arbitrary medium (e.g., text terminal, video game environment). The interrogator is informed that one subject is human and the other is a computer but the true identities of the subjects remains unknown to him. After spending some time with the subjects, the interrogator is asked which is human and which is a computer. The computer is said to have passed the test if it is incorrectly labeled as a human, effectively fooling the interrogator by displaying human-like behaviour in the medium. The Turing test is used for evaluating believability in some high-profile competitions. For example, the Loebner Prize is awarded to programs that excel at natural language processing [32] while the 2K Bot Prize tests the realism of combat decisions made by FPS bots [44].

The Turing test is very challenging to pass because the interrogator (or "judge") is permitted to interact with the subjects through any means afforded by the medium, possibly over long periods of time. Unfortunately, even if a computer's performance can be described as human-like, a random decision will be made by the judge if he cannot determine which subject is human. While this issue could be resolved by allowing judges to label subjects by a third "uncertain" option, there are several other ways for grading the believability of a computer program.

Sometimes judges may only be permitted to observe subject behaviour, typically due to the limitations of the chosen medium. In these cases, judges are shown video clips of the subjects in action and are then asked to grade the believability of each. Instead of judging a collection of clip simultaneously, Mac Namee asks judges to compare pairs of clips, stating one as "more believable" than the other [33]. This approach facilitates a partial ordering of clips when several pairs are compared. However, two judges may have conflicting opinions and grade the same pair of clips differently. Moreover, every pair of clips must be evaluated to construct a complete ordering.

Another method for grading believability is to assign a single representative value to a clip. Gorman et al. rate video clips of Quake II bot behaviour on a five-point scale, 1 being "definitely artificial" and 5 being "definitely human" [18]. These values are then adjusted according to the judge's experience level and averaged over all gradings provided by all judges. The final result is a *believability index* in the range (0, 1), defined by the authors as "a weighted representation of the degree to which a given type of clip was regarded as human". Alternatively, Laird and Duchi use a combination of both an objective measure and the Turing test: judges are asked to grade the believability of a bot on a scale of 1 to 10 and provide a guess as to whether the bot is human [27].

Because the process of judging bot behaviour is time-consuming and requires human experts, a formal definition of believability would be ideal to automate the grading of bot performances. However, Gorman et al. note that no such definition currently exists [18]. Livingstone proposes

a set of believability criteria, divided into three categories: Plan, Act, and React [31]. However, these criteria have no objective specification (e.g., "AI should react to presence of foes and allies") and are by no means complete. A model of human behaviour could be used to construct a general believability measure in the same way that human predictions help form the human similarity error in Section 2.3.2. However, it may only be practical to build approximations of such a model, given that specifying a complete, explicit description of human behavior is a difficult task.

Chapter 4

Background

This chapter presents background knowledge on dynamic state estimation, a general problem formulation for modeling the state of a system over time (Section 4.1). Section 4.2 introduces the Bayesian filter, a solution to dynamic state estimation that relies on several key assumptions about the problem domain. These assumptions allow specialized algorithms to provide approximations of the system's current state given all information available about the system. Hidden Markov models and particle filters are two such algorithms, both of which are presented in Section 4.3 and Section 4.4 respectively. The chapter concludes with a discussion of the advantages each algorithm offers and the tradeoffs necessary to maintain these advantages.

4.1 Dynamic State Estimation

The *dynamic state estimation* (DSE) problem is concerned with modeling the state of a system over time. We present the discrete-time formulation of the DSE problem; that is, the system is assumed to change states only at discrete intervals or *time steps*. On each time step t, the system transitions to state S_t from its previous state S_{t-1} and an observation O_t is emitted by the system (Figure 4.1). The history of the system's states can be written out as the sequence $(S_1, ..., S_t)$, which shall be denoted by $S_{1:t}$ (a similar sequence can be constructed for the history of observations). Note that S_1 corresponds to the system's initial state at which time O_1 is emitted.

Dynamic state estimation is a challenging problem because it prohibits the state S_t from being determined directly. Instead, one must *infer* S_t using any information known about the system. Because there may be a measure of uncertainty when making such inferences, it is common to use a probability distribution $P(S_t)$ to represent the current state of the system. Using this notation, the probability $P(S_t = s)$ denotes the likelihood of the system being in state s at time t.

4.2 Bayesian Filtering

One of the common questions asked about the DSE problem is "What is the current state of the system?". This is answered by calculating $P(S_t \mid O_{1:t})$, the posterior distribution over all possible

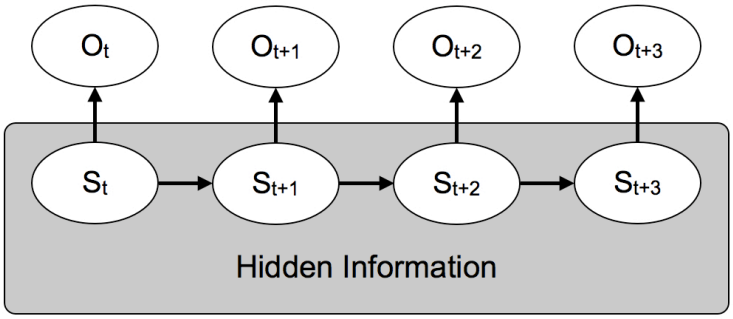


Figure 4.1: Bayesian network structure for a hidden Markov model.

states given all the evidence (i.e., the entire history of observations). This inference task can be achieved using *Bayesian filtering*. Bayesian filtering makes two assumptions based on the Markov property: 1) the current state of the system is conditionally independent given the previous state (i.e., $P(S_t \mid S_{1:t-1}) = P(S_t \mid S_{t-1})$), and 2) the current observation emitted is conditionally independent given the current state (i.e., $P(O_t \mid S_{1:t}, O_{1:t-1}) = P(O_t \mid S_t)$). The standard Bayesian filtering equation (Equation 4.5) can be derived using these assumptions [55]:

$$
\begin{aligned}
P(S_t \mid O_{1:t}) &= P(S_t \mid O_t, O_{1:t-1}) && (4.1)\\
&= \eta\, P(O_t \mid S_t, O_{1:t-1})\, P(S_t \mid O_{1:t-1}) && (4.2)\\
&= \eta\, P(O_t \mid S_t)\, P(S_t \mid O_{1:t-1}) && (4.3)\\
&= \eta\, P(O_t \mid S_t) \int_{S_{t-1}} P(S_t \mid S_{t-1}, O_{1:t-1})\, P(S_{t-1} \mid O_{1:t-1}) && (4.4)\\
&= \eta\, P(O_t \mid S_t) \int_{S_{t-1}} P(S_t \mid S_{t-1})\, P(S_{t-1} \mid O_{1:t-1}) && (4.5)
\end{aligned}
$$

Note that η is a normalizing constant. Equation 4.1 is derived by splitting up the evidence $O_{1:t}$. Bayes' Rule is used to produce Equation 4.2, which is then simplified to Equation 4.3 by using the second assumption. Equation 4.4 illustrates the marginalization over S_{t-1} and simplified again by the first assumption to produce Equation 4.5. The Bayesian filter computes $P(S_t \mid O_{1:t})$ by modifying the distribution of the previous time step $P(S_{t-1} \mid O_{1:t-1})$, a process called *recursive estimation*. Hereafter $\alpha_t(S) = P(S \mid O_{1:t})$ will be referred to as a *message* that is updated on each time step, representing the estimation of the system's current state.

In general, the Bayesian filter can be thought of as a two-step process. First, the previous message $\alpha_{t-1}(S_{t-1})$ is modified by the *transition function* $P(S_t \mid S_{t-1})$, accounting for the state transition of the system. Second, the message is updated by the *observation function* $P(O_t \mid S_t)$, effectively weighting the message by the probability of observing O_t. Given a *prior* $P(S_1)$, these two

steps can be used repeatedly to compute $\alpha_{1:t}(S_t)$, all the messages from time 1 to time t.

There are many other formulations of the DSE problem (and hence the Bayesian filtering equation), each of which is suited to different problem domains. For example, an entity external to the system may enact an action A_t on each time step, which has the potential to affect the system transitions. These actions can be incorporated into the transition function by $P(S_t \mid S_{t-1}, A_t)$. However, for the purposes of this work, no such variables are considered to be a part of the DSE problem.

4.3 Hidden Markov Models

As presented, Bayesian filtering requires the prior, transition function, observation function, and messages to be integrable in order to compute the result of Equation 4.5. This formulation is satisfactory for Kalman filters where each function and message are mulitvariate Gaussian distributions [23]. However, Gaussian distributions are easily represented by a closed formula. For systems that do not have closed forms of their functions and messages, a common solution is to discretize the state space. The Bayesian filter resulting from this modification is called the *Hidden Markov Model* (HMM). Equation 4.6 presents the update expression for HMMs, a discrete version of Equation 4.5.

$$\alpha_t(S) = \eta\, P(O_t \mid S) \sum_{S'} P(S \mid S')\, \alpha_{t-1}(S') \tag{4.6}$$

Implementing the update as a sequential procedure results in Algorithm 1, the HMM update function. Note that lines 1 and 2 correspond to the two-step process mentioned in Section 4.2.

Algorithm 1 Hidden Markov model update function

Require: Messages $\alpha_{1:t-1}$
Require: Observation O_t
1: $\alpha_t(S) \leftarrow \sum_{S'} P(S \mid S')\, \alpha_{t-1}(S')$
2: $\alpha_t(S) \leftarrow \eta\, P(O_t \mid S)\, \alpha_t(S)$
3: **return** α_t

4.3.1 Hidden Semi-Markov Models

In some problem domains, it is necessary for the system to remain in a state s for a finite length of time. Consider the problem of tracking a car on city streets where the car's state could be defined by its GPS coordinates. An HMM should account for the situations where the car is waiting at a red stop light. One solution is to allow the transition function to include self-transitions for s (i.e., $P(S_t = s \mid S_{t-1} = s) > 0$). Unfortunately, a caveat of this decision is that the probability $\alpha_t(S_t = s)$ diminishes asymptotically to 0 over time; in effect, the message permanently maintains the possibility that the system could remain in s indefinitely.[1] Moreover, the probability of transi-

[1] Except in the case where the observations prohibit the system from occupying s (i.e., $P(O_t \mid S_t = s) = 0$).

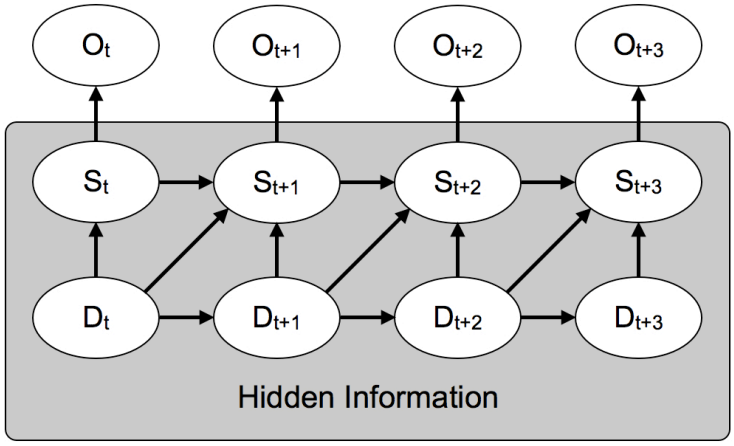


Figure 4.2: Bayesian network structure for a hidden semi-Markov model.

tioning from s is the same on every time step. It would be undesirable for the HMM to assume the vehicle could move at any time step while the car proceeds only when the light turns green. A new approach must be considered that models the time spent in each state.

Hidden Semi-Markov Models (HSMM) are similar to HMMs with the exception that the system is allowed to change states within $\hat{d} \in \mathbb{Z}^+$ times steps. A counter D is used to represent the duration of the system's current state. On each time step, D is decremented until it reaches 0 at which point the system changes states and D is "reset" according to a *duration function* $P(d \mid S_t)$. Figure 4.2 illustrates how the counter can be incorporated into the HSMM structure. The dependencies between the counter and state nodes, represented by arrows in Figure 4.2, are defined by the following equations [49]:[2]

$$P(S_t = s \mid S_{t-1} = s', D_{t-1} = d) = \begin{cases} \delta(s, s') & \text{if } d > 0 \text{ (remain in same state)} \\ P(S_t = s \mid S_{t-1} = s') & \text{if } d = 0 \text{ (change states)} \end{cases} \tag{4.7}$$

$$P(D_t = d \mid S_t = s, D_{t-1} = d') = \begin{cases} \delta(d, d' - 1) & \text{if } d' > 0 \text{ (decrement counter)} \\ P(d \mid S_t = s) & \text{if } d' = 0 \text{ (reset counter)} \end{cases} \tag{4.8}$$

To ensure that $\alpha_{1:t}(S)$ can be recursively estimated, another Markov assumption must be made about the system durations: 3) the duration of the system's state is conditionally independent given the current state (i.e., $P(d \mid S_{1:t}, d_{1:t}) = P(d \mid S_t)$). This third assumption allows us to derive Equation 4.13, the update expression for HSMMs.

[2]The Kronecker delta function, denoted $\delta(s, s')$, evaluates to 1 when $s = s'$ and 0 otherwise.

$$\alpha_t(S,d) = P(O_{t-d+1:t} \mid S,d) \sum_{S'} \sum_{d'} P(S,d \mid S',d')\, \alpha_{t-d}(S',d') \tag{4.9}$$

$$= P(O_{t-d+1:t} \mid S,d) \sum_{S'} \sum_{d'} P(S \mid S')\, P(d \mid S)\, \alpha_{t-d}(S',d') \tag{4.10}$$

$$= P(d \mid S)\, P(O_{t-d+1:t} \mid S,d) \sum_{S'} P(S \mid S') \sum_{d'} \alpha_{t-d}(S',d') \tag{4.11}$$

$$\alpha_t(S) = \sum_{d} P(d \mid S)\, P(O_{t-d+1:t} \mid S,d) \sum_{S'} P(S \mid S')\, \alpha_{t-d}(S') \tag{4.12}$$

$$= \sum_{d} P(d \mid S) \left[\prod_{u=t-d+1}^{t} P(O_u \mid S) \right] \sum_{S'} P(S \mid S')\, \alpha_{t-d}(S') \tag{4.13}$$

Equation 4.9 shows how the HMM update procedure (Equation 4.6) can be extended to represent the joint probability of S and d [49]. Using the third assumption, it can be asserted that $P(S,d \mid S',d') = P(S \mid S')P(d \mid S)$ (Equation 4.10). Equation 4.11 rearranges terms and summations for simplicity. Because $\alpha_t(S) = \sum_d \alpha_t(S,d)$, the right-hand side of Equation 4.12 is marginalized by d. Finally, the second assumption allows for $P(O_{t-d+1:t} \mid S,d)$ to be rewritten as a 6product of observations, resulting in Equation 4.13. This equation forms the basis for Algorithm 2, the HSMM update function.

Algorithm 2 Hidden semi-Markov model update function

Require: Message α_{t-1}
Require: Partial Messages $\beta_{t-\hat{d}+1:t-1}$
Require: Observations $O_{t-\hat{d}:t}$
1: $\beta_t(S_t) \leftarrow \sum_{S_{t-1}} P(S_t \mid S_{t-1})\, \alpha_{t-1}(S_{t-1})$
2: $A \leftarrow z(|S|)$
3: $B \leftarrow P(O_t \mid S_t)$
4: $C \leftarrow o(|S|)$
5: $m \leftarrow \min(\hat{d}, t)$
6: **for** $d = 1$ to $m-1$ **do**
7: $\quad A \leftarrow A + [P(d \mid S_{t-d}) \cdot B \cdot \beta_{t-d+1}(S_{t-d+1})]$
8: $\quad B \leftarrow B \cdot P(O_{t-d} \mid S_{t-d})$
9: $\quad C \leftarrow C - P(d \mid S_{t-d})$
10: **end for**
11: $\alpha_t(S_t) \leftarrow \eta\,(A + [C \cdot P(O_{t-m} \mid S_{t-m}) \cdot \beta_{t-m+1}(S_{t-m+1})])$
12: **return** $\alpha_t(S_t)$

Note that $z(n)$ is a vector of zeroes and $o(n)$ is a vector of ones, both with size n. The hidden semi-Markov model update begins by computing the partial message $\beta_t(S_t)$ on line 1. This message is equivalent to the previous full message $\alpha_{t-1}(S_{t-1})$ multiplied by the transition function to account for opponent movement (see Equation 4.14). However, $\beta_t(S_t)$ does not yet account for observations and the values it contains are not normalized. Algorithm 2 relies on a history of $\hat{d}$ partial messages, which can be constructed and stored on each update in a manner similar to that of the full messages (that is, $\beta_t(S_t)$ is computed when $\alpha_t(S_t)$ is computed).

$$\beta_t(S_t) = \sum_{S_{t-1}} P(S_t \mid S_{t-1})\, \alpha_{t-1}(S_{t-1}) \tag{4.14}$$

On lines 2 through 5, several vectors are initialized to either ones or zeros and a threshold m is set to either the largest possible duration value or the current time step, whichever is less. Line 6 begins the main loop that incrementally builds the posterior distribution from the past m partial messages. Line 7 updates the vector A, which serves as an intermediate computation of the posterior. Specifically, A is incremented by the partial message generated at time $t - d + 1$, multiplied by both B and the probability of transitioning after d time steps. Vector B acts as a filter, accumulating observation probabilities at line 8 on each iteration of the loop. In this way, if the system were to wait d time steps before transitioning, any estimation must be filtered by all the observations that were emitted within the past d time steps. On line 9, vector C records the duration probabilities. After the loop has finished processing, C will contain any "remaining" duration values that were not applied to A on line 7. Finally, line 12 stores the complete posterior distribution in $\alpha_t(S)$ using C as the probability of transitioning after m time steps.

4.4 Particle Filters

The defining attributes of hidden Markov models are that the state space is assumed to be both discrete and finite and that the system changes states according to a Markovian transition policy. If the system does operate according to these conditions, then the HMM provides an optimal solution to the DSE problem. However, for systems that do not, some error may be introduced in the messages. Recall the car example and assume that the HMM models system states as individual city streets. Although the car can be positioned at many different GPS coordinates along a given street, a single probability represents the likelihood of the car driving anywhere on the street. Modeling states at the street level can incur some error. For example, though the street probability may be large, the likelihood of the car occupying lanes with oncoming traffic should be close to 0. This error can be mitigated by reducing the state size at the cost of increasing the number of states required to cover the city. However, the error may only be eliminated completely by modeling an infinite number of states.

An alternative to the HMM is the *particle filter* (PF), a model that represents the message α_t by a set of weighted particles $X = \{(w^i, p^i), i = 1, ..., n\}$ where weights conform to $\sum_i w^i = 1$ and particles p^i are scalar vectors. These particles are an approximation (specifically, a Monte Carlo sampling) of the continuous form of the message. In this way, a continuous message can be achieved by increasing the number of particles n towards infinity (Equation 4.15).

$$P(S_t \mid O_{1:t}) = \lim_{i \to \infty} \sum_i w_t^i\, \delta_{S_t - p_t^i} \tag{4.15}$$

While there are many types of PFs, each tailored to particular assumptions made about the domain, this work uses a general variant: *sampling importance resampling* (SIR).[3] An SIR particle filter uses two techniques to generate a distribution of particles representing α_t, both of which are presented in the following subsections.

4.4.1 Importance Sampling

On each time step t, the goal of the particle filter update process is to produce a set of n particles that approximates the message α_t. The first solution that may come to mind is to sample particles directly from α_t. Unfortunately, this approach is rarely used in practice as it may be computationally intensive to sample from α_t.

Importance sampling is a technique used to estimate $E_{Q(x)}f(x)$ where $f(x)$ is a random variable and $Q(x)$ is a distribution that should not (or cannot) be sampled directly. Instead, points can be sampled from a known distribution $R(x)$, referred to as the *proposal distribution*, and then correcting the results by *importance weights* $w(x) = \frac{Q(x)}{R(x)}$. This method is shown to be sound by Equation 4.19, which presents the equality of $E_{Q(x)}f(x)$ and $E_{R(x)}f(x)\,w(x)$.

$$E_{Q(x)}f(x) = \int_x f(x)\,Q(x) \tag{4.16}$$

$$= \int_x f(x)\,\frac{Q(x)}{R(x)}\,R(x) \tag{4.17}$$

$$= \int_x f(x)\,w(x)\,R(x) \tag{4.18}$$

$$= E_{R(x)}f(x)\,w(x) \tag{4.19}$$

The SIR particle filter defines $Q(x) = \alpha_t = P(p_t \mid O_{1:t})$, the distribution to be represented by sampled particles. By expanding $Q(x)$ to the full Bayesian filter form (Equation 4.5), the proposal distribution is chosen to be $R(x) = P(p_t \mid p_{t-1})$, leaving the weights to be defined as $w_t^i = P(O_t \mid p_i^i)$ [3].[4] Thus, given a previous set of particles X_{t-1} sampled from α_{t-1}, an approximation of $Q(x)$ can be generated by sampling particles from the transition function $P(p_t \mid p_{t-1}^i)$ and weighting them by the observation function $P(O_t \mid p_t^i)$.

4.4.2 Resampling

After several updates, a phenomenon called *degeneracy* often occurs where a few particles accumulate relatively large weights compared to the others. Maintaining a majority of particles with low weights is undesirable because the particle filter is reduced to a select few possibilities with any degree of "importance". To combat this trend, an SIR particle filter resamples all particles (with

[3] See the tutorial by Arulampalam et al. for an overview of other particle filter variants [3].

[4] Although it is possible for both the proposal distribution and importance weights to be switched, the choices presented here are standard practice for SIR particle filters.

replacement) from the new weighted distribution and the weights are set to uniform probabilities. This process redistributes outliers around particles with high weights, thus providing a more accurate representation of α_t.

Algorithm 3 shows the implementation of an SIR particle filter.[5] Note that lines 1 through 6 correspond to importance sampling process while lines 7 through 11 resample particles.

Algorithm 3 SIR Particle Filter update function

Require: A set of particles X_{t-1}
Require: Observation O_t
1: $\hat{X}_t \leftarrow 0$
2: $X_t \leftarrow 0$
3: **for** $i = 1$ to $|X_{t-1}|$ **do**
4: sample $p_t^i \sim P(p_t \mid p_{t-1}^i)$
5: $w_t^i \leftarrow P(O_t \mid p_t^i)$
6: $\hat{X}_t \leftarrow \hat{X}_t \bigcup \{(p_t^i, w_t^i)\}$
7: **end for**
8: **for** $i = 1$ to $|\hat{X}_t|$ **do**
9: sample $p_t^i \sim \hat{X}_t$
10: $w_t^i \leftarrow (|\hat{X}_t|)^{-1}$
11: $\hat{X}_t \leftarrow \hat{X}_t \bigcup \{(p_t^i, w_t^i)\}$
12: **end for**
13: **return** X_t

4.5 Hidden Markov Models versus Particle Filters

Although both hidden Markov models and particle filters are solutions to the DSE problem, each technique has its own advantages and disadvantages. As mentioned in Section 4.3, hidden Markov models assume that the state space is discrete and finite. While this simplification results in a compact solution to the DSE problem, the computation time required to update HMMs increases with the number of states. Thus for large domains HMMs become prohibitively expensive to compute.

In contrast, particle filters can model a distribution with potentially greater fidelity than that of HMMs because they do not make the same assumptions about the state space. PFs can also be adapted to associate particles with a duration counter similar to that of HSMMs, which will be explained in Chapter 5. Nominatively, the computation costs of PF updates scale only with the number of particles, as opposed to the number of system states. Yet, there are two drawbacks which must be noted. First, a large number of particles must be maintained by a PF to accurately represent complex distributions with many state variables. Second, while resampling does help reduce degeneracy, it comes at the cost of repositioning outliers close to other particles (a phenomenon called *sample impoverishment*). Both of these issues can increase the time required to run particle filter update function.

[5] The notation w_t^i and p_t^i corresponds to the appropriate values of particles in X_t.

Chapter 5

Proposed Approach

This chapter presents the primary contribution of this thesis: the design and implementation of predictors. These predictors are based on the hidden semi-Markov model and particle filter techniques described in Chapter 4, which are herein simply referred to as *motion models*. The following section discusses the general structure of a predictor and explains how motion models are incorporated into this structure. Section 5.3 details how the prior, transition, and duration functions for models can be learned from game log data. As well, this section also describes a method used to compute the values of the observation function online. Finally, Section 5.4 explains how models are managed during the course of a game, particularly when special events occur such as the sighting of an opponent and the announcement of a player being eliminated from the game.

5.1 Joint and Factored Models

To use hidden semi-Markov models and particle filters for predicting opponent positions, the prediction problem must be formulated in terms of a discrete DSE problem. Conceptually, a model should always maintain an estimation of each opponent's position even as opponents move throughout the game environment. This can be achieved by treating opponents as the system of the DSE problem. Because a system must always reside in one of a finite number of states, the game environment is partitioned into $H = (h_1, h_2, ..., h_i)$ regions and each opponent is required to occupy a single region at all times.

There are two common ways to represent opponent positions by motion models. One method is to estimate the *joint position* of all opponents. This technique uses a single model whose state space is defined by H^n (where n is the number of opponents alive), essentially the Cartesian product of all regions occupied by opponents. A joint-position model has strong representational power because synchronization among enemy movements can be encoded in the transition and duration functions. In effect, this allows strategies and tactics involving more than one player to be learned from gameplay data. Unfortunately, the state space grows exponentially with the number of opponents being tracked, a drawback that hinders the practicality of this technique. This drawback becomes prevalent

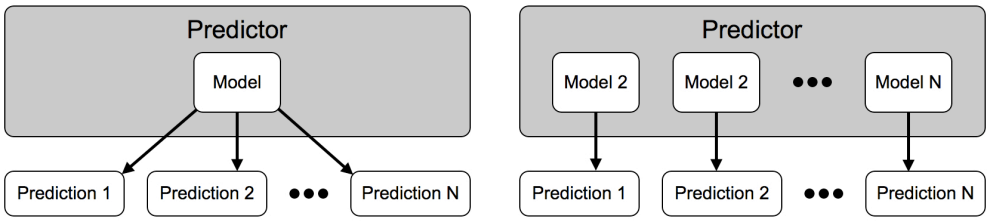


Figure 5.1: Structural differences between joint (left) and factored (right) predictors. Predictions 1 through N define the full prediction vector.

when specifying the transition function. For a modest partitioning of 100 regions and tracking 5 opponents, there are 10^{19} probabilities that must be stored when using a tabular format, an intractable task for any current computer. Even if sufficient memory were available, a significant amount of training data would be required to learn these probabilities and any calculation involving the whole transition function could not be computed in an online scenario.

The alternative method is to use a *factored* representation of opponent positions. This technique uses several models to track opponents in a game, namely one motion model per opponent alive. In contrast to the joint representation, the state space of each factored model is defined by H. Unfortunately, the transition and duration functions in this representation are unable to capture any associations between enemy movements as each model is completely independent of the others. However, the computations required to produce a prediction are tractable and comparatively little data is needed for training due to the significantly reduced state space. Figure 5.1 illustrates the structural differences between predictors using these two different formulations. The factored representation is adopted for predictors in this work.

5.2 Predictor Anatomy

A predictor is responsible for maintaining an estimate of each opponent's position. By using a factored state representation, the predictor need only manage the motion models themselves; the task of updating an opponent position is contained within an individual model and is abstracted from the predictor. On each time step, a predictor performs four key operations in the following order:

1. Deallocate models to ensure that the total number is equal to the number of opponents alive.
2. Associate each sighting of an opponent (if any) with a unique model.
3. Update all models to the current time step.
4. Generate a prediction from each model.

The implementation of the first two operations are independent of the model type being used (be it HSMMs or PFs) and are discussed in Section 5.4. In contrast, the third and fourth operations are particular to each model type and are detailed in the following subsections.

5.2.1 Hidden Semi-Markov Model Configuration

To utilize an HSMM for predicting an opponent's position, the map of the game environment is discretized via a two-dimensional grid G. By setting $H = G$, each grid cell is considered a state of the system (i.e., a possible location for an opponent to occupy). The likelihood of the opponent occupying grid cell $g \in G$ at time t is determined by calculating $P(S_t = g \mid O_{1:t})$. Note that for a large partitioning of a game map, there may be many places for an opponent to reside within a single grid cell. Therefore an HSMM's prediction is the coordinate pair corresponding to the center of the grid cell with the highest probability, or more formally $\arg\max_{g \in G} P(S_t = g \mid O_{1:t})$.

5.2.2 Particle Filter Configuration

The formulation of the particle filter in Section 4.4 cannot be applied directly to the prediction problem; the proposal distribution and importance weights as presented are undefined. Darken and Anderegg propose a configuration of a particle filter for predicting opponent positions [15]. This work implements their method and extends it by incorporating the use of training data and allowing players to be stationary for several time steps. Specifically, the prior, transition function, duration function, and grid required by HSMMs are used as a basis for constructing the proposal distribution and importance weights.

Unlike an HSMM, a particle filter does not depend on the set of possible system states to be finite. Thus, setting $H = \mathbb{R}^2$ enables each particle to act as a candidate coordinate of the opponent's position. Intuitively, these particles should be moved about the map according to the motion models. Algorithm 4 shows the update process for generating a new set of particles from the particles of the previous time step. Let $C_G(p) \in G$ represent the grid cell that contains particle p. On each time step t, current particle p_t^i is produced by adding a *movement vector* $m^i \in \mathbb{R}^2$ to previous particle p_{t-1}^i (line 7). The vector is constructed from two pieces of information: a target $q^i \in \mathbb{R}^2$, the center of a grid cell sampled from the transition function (line 3), and a counter d^i sampled from the duration function (line 4). Note that m^i is recalculated with a new target and velocity on time steps where p_t^i enters the grid cell of target q^i (line 2). Each importance weight w^i is calculated as the probability of observing O given that opponent is indeed located at p_t^i (line 8). Finally, the resampling phase of the particle filter takes place on lines 10 through 15 with the function returning the new set of particles Y as well as updated movement vectors and targets.

A particle filter's prediction is generated by randomly selecting a particle. Although this may appear to be a naïve selection mechanism, this policy is chosen to contrast with the HSMM prediction policy. A PF prediction reflects the variance of the posterior distribution of particles. There

Algorithm 4 Particle Filter

Require: A set of weighted particles X
Require: A set of targets Q
Require: A set of movement vectors M
Require: Observation O
Ensure: $|X| = |Q| = |M|$
for $i = 1$ to $|X|$ **do**
  if $C_G(p^i) = C_G(q^i)$ **then**
    sample $q^i \sim P(S_t \mid S_{t-1} = C_G(p^i))$
    sample $d \sim P(D \mid S_t = C_G(q^i))$
    $m^i \leftarrow (q^i - p^i)d^{-1}$
  end if
  $p^i \leftarrow p^i + m^i$
  $w^i \leftarrow P(O \mid S_t = C_G(p^i))$
end for
for $i = 1$ to $|X|$ **do**
  sample $p^j \sim X$
  $q^i \leftarrow q^j$
  $m^i \leftarrow m^j$
  $Y \leftarrow Y \bigcup \{(p^j, |X|^{-1})\}$
end for
return Y, Q, M

is a good chance the prediction will be selected from areas of the map with high concentrations of particles and less so in outlying regions. The effectiveness of both the HSMM and PF prediction methods are evaluated in Chapter 6.

5.3 Model Training

Chapter 3.3 proposed that perception reasoning and opponent modeling can be used to inform a predictor's estimates of opponent positions. This section discusses how these two techniques can be incorporated into motion models, specifically by encoding expert human gameplay in the transition and duration functions and by representing the sensory information perceived by friendly players in the observation functions.

5.3.1 Building the Prior

Every motion model uses a common prior particular to a given map and team. Given the set of possible starting locations for opponents $K = \{k \mid k \in \mathbb{R}^2\}$, the prior is constructed as a distribution over G as defined by Equation 5.1.

$$P(S_1 = g) = \frac{\sum_k \delta(C_G(k), g)}{|K|} \tag{5.1}$$

5.3.2 Building the Transition and Duration Functions

The transition and duration functions are used to encode the movement directions and velocities of opponents respectively. Both of the functions can be learned simultaneously by analyzing player trajectories from several game logs. Game logs store player histories as a set of game state "snapshots" or *frames*; that is, at certain time intervals the world coordinates, view cone, health, and other attributes are recorded for each player. This format provides a convenient way to update motion models: time steps are fixed to occur with the same frequency at which frames are recorded. By varying the rate at which game frames are recorded, motion models can be adjusted to take advantage of the amount of computation power available.

A player's trajectory is defined as the sequence of map coordinates $J = \{j_1, j_2, ..., j_n\}$ that the player occupies on each game frame. Algorithm 5 iterates over all trajectories in the game logs to build the transition and duration functions, represented in a tabular format as matrices T and D respectively. The algorithm keeps track of the previous grid cell a player occupies as the game progresses. On frame t, if a player is observed to reside in the same cell as on the previous frame (i.e., $j_t = j_{t-1}$), a counter d is incremented by 1 (lines 9 through 14). When the player transitions to a different cell (i.e., $j_t \neq j_{t-1}$), the values $T[j_t, j_{t-1}]$ and $D[d, j_{t-1}]$ are incremented and d is reset (lines 15 and 16). Once all trajectories have been analyzed, both T and D are normalized resulting in the transition and duration functions respectively (lines 18 through 21).

5.3.3 Building the Observation Function

The observation function is used to represent the possible areas of the map visible to members of the friendly team on the current time step. Because these areas change as players move around on the map, the observation function must be constructed online. To make this task tractable, the game environment is discretized via a three-dimensional grid V. Binary values denoting a clear line-of-sight (LOS) between the center coordinates of each pair of visibility cubes are computed offline. The visibility grid is used as a lookup table to quickly determine the areas of the map visible from a given location. The use of V for LOS calculations introduces some error (e.g., not all parts of cube may share the same LOS as its center coordinate) which can be mitigated by decreasing the size of cubes. However, doing so will increase the memory required by the lookup table.

The process of constructing the observation function begins by determining the set of cubes $W \subseteq V$ visible to at least one player on the friendly team. There are three steps involved in computing W which is repeated for each friendly player in the game (an example is shown in Figure 5.2):

1. Identify $C_V(h)$, the visibility cube containing a friendly's head located at coordinates h.
2. Determine the set of visibility cubes W' with unobstructed line-of-sight to $C_V(h)$.
3. Filter W' to elements that are within the player's view cone. These elements compose W.

Algorithm 5 Transition and Duration Construction

Require: A set of game logs L
Require: Map grid G
Require: Transition matrix T
Require: Duration matrix D
Ensure: $T[x, y] = 0, \forall x, y$
Ensure: $D[x, y] = 0, \forall x, y$

for all game logs l in L **do**
  for all player trajectories J in l **do**
    $n \leftarrow |J|$
    $f \leftarrow 1$
    $g_{cur} \leftarrow C_G(j_f)$
    increment f
    while $(f \leq n)$ **do**
      $d \leftarrow 0$
      repeat
        $g_{prev} \leftarrow g_{cur}$
        $g_{cur} \leftarrow C_G(j_f)$
        increment d
        increment f
      until $(g_{cur} \neq g_{prev}$ or $f > n)$
      increment $T[g_{cur}, g_{prev}]$
      increment $D[d, g_{cur}]$
    end while
    for all g in G **do**
      normalize $T[x, g]$ over x
      normalize $D[x, g]$ over x
    end for
  end for
end for
return T, D

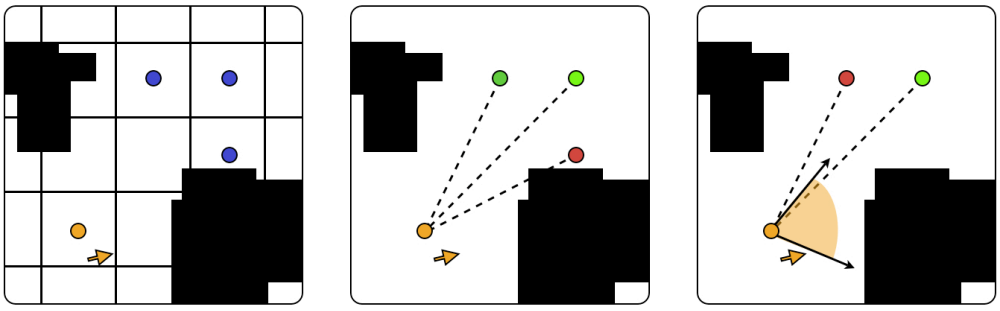


Figure 5.2: Example computing W. Left: A friendly player (orange arrow) is mapped to the containing visibility cube (step 1). Center: The green centers are visible while the red center has an obstructed LOS (step 2). Right: The green center is within the player's field-of-view (step 3).

Once W is known, the observation function can be defined by mapping the visibility cubes in W to their respective grid cells. Let $X(g)$ represent the set of all visibility cubes whose centers (disregarding the z-coordinate) are contained in grid cell g. Equation 5.2 shows how $X(g)$ is used to evaluate $P(O_t \mid S_t = g)$. Note that this process requires the dimensions of the grid cells to be larger or equal to the dimensions of the visibility cubes.[1]

As well, Equation 5.2 calculates the weights for particles using the method described above (effectively line 8 of Algorithm 4). However, in the event that the dimensions of grid cells and visibility cubes are not equal, another two-dimensional grid G' with the same cell sizes as V can be substituted for G. This ensures that only the elements of V that are directly above and below a grid cell $g' \in G'$ are considered when $X(g')$ is evaluated. All experiments in Chapter 6 use this enhancement for particle filters.

$$P(O_t \mid S_t = g) = 1 - \frac{|W \cap X(g)|}{|X(g)|} \tag{5.2}$$

$$w_t^i = P(O_t \mid S_t = C_G(p_t^i)) \tag{5.3}$$

$P(O_t \mid S_t)$ is a vector of size $|G|$ with each value bounded in the range $[0, 1]$. Intuitively, each value represents the portion of a corresponding grid cell in G that is visible to the friendly team. For example, if grid cell g is completely observed, then $P(O_t \mid S_t = g) = 0$ denoting that it is impossible for an opponent to reside in g. Motion models reflect this by weighting intermediate probability distributions by the observation function. Consider Equation 5.4, the discrete Bayesian filter reprinted from Section 4.3. With regards to the above example, $\alpha_t(S = g)$ evaluates to 0 and therefore no prediction should be made at g.

[1] By allowing the dimensions of elements in V and G to be independent, one can control sizes of the visibility cubes to be as small as possible within practical limits. A finer visibility grid produces more accurate LOS results.

$$\alpha_t(S = g) = \eta\, P(O_t \mid S = g) \sum_{S'} P(S = g \mid S')\, \alpha_{t-1}(S') \tag{5.4}$$

5.4 Model Management

The previous sections in this chapter discussed how motion models update their distributions and the processes for generating predictions. The predictor executes these operations across all models simultaneously, ensuring that the time steps of all models are synchronized. Occasionally, an event occurs at which time special updates must be made to a particular model. Specifically, when an opponent is sighted by a friendly player, the predictor must match that sighting to a model and update it accordingly. Similarly, when an opponent is eliminated from the game, a model must be chosen and removed to maintain the one-to-one ratio of models to opponents. The policies for dealing with these two situations are presented below.

If an opponent is sighted in a grid cell $g \in G$ at time t, the posterior distribution of the motion model tracking that opponent is explicitly adjusted such that $P(S_t = g \mid O_{1:t}) = 1$ and $P(S_t \neq g \mid O_{1:t}) = 0$. This constrains the distribution to the grid cell containing the opponent. However, which model should receive this update? The prediction problem assumes that all opponents are indistinguishable from each other thus prohibiting sightings from being directly associated with a model. The approach used in this work is to update the model that has the "greatest likelihood" of sighting the opponent at his current location; that is, the model with the highest value for $P(S_t = g \mid O_{1:t-1})$ is matched and all others update normally.

One drawback to this policy is that it is possible for a single model to be matched with sightings of two different opponents on different time steps. While this is undesirable (ideally two separate models should receive updates), it is not an illogical matching. A model is matched with two (or more) different sightings only when the same sequence of sightings could have been produced by a single opponent. Unfortunately, an incorrect matching can have lasting consequences on the quality of a model's predictions. Consider the worst case where only single sightings are observed and every sighting is assigned to a single predictor. The remaining predictors no longer receive any special updates and as a result their distributions begin to become stationary. The maximum probability in such a distribution would be very low, a situation that could be avoided had the sightings been divided among the predictors with another matching policy. The effects of different matching policies on predictor performance would be worth investigating in future research.

If 2 or more opponents are sighted on the same time step, the assignment process is expanded to match the grid cells $(g^1, ..., g^n)$ containing sighted opponents to motion models. The chosen matching is determined by Equation 5.5, specifically the one that results in the maximal sum of $P(S_t = g^i \mid O_{1:t-1})$ where $\pi(i)$ is a permutation over the motion models. Note that ties are broken randomly when calculating the summations. Thus, for a single sighting in grid cell g where

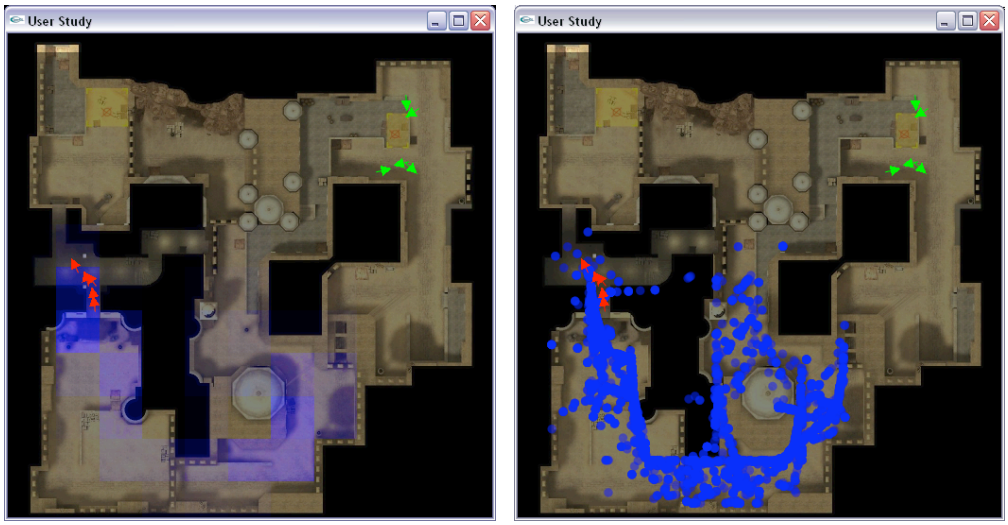


Figure 5.3: The internal representations of predictors using both hidden semi-Markov models (left) and particle filters (right). Green arrows represent friendly players while the red arrows represent opponents. The blue hue squares in the left image illustrate the posterior distribution; the brighter the colour, the greater the likelihood of an opponent occupying the square. The blue dots in the right image denote individual particles.

the probability $P(S_t = g \mid O_{1:t-1}) = 0$ for each predictor, the sighting is assigned to a random predictor.

$$\arg\max_{\pi} \sum_{i=1}^{n} P(S_t = g^i \mid O_{1:t-1})^{\pi(i)} \tag{5.5}$$

Finally, when an opponent is eliminated from the game, models in a similar manner by matching the models to the last grid cells occupied by dead opponents. A model matched at this stage is removed from the predictor for the remainder of the game.

5.5 Summary

This chapter detailed how hidden semi-Markov models and particle filters can be configured to predict opponent positions. A graphical representation of these models in action is shown in Figure 5.3. The posterior distribution of the HSMM is depicted by blue hue squares, which are few in number but represent general areas of the map. In contrast, the distribution of the particle filter is approximated by numerous blue dots that are not confined to the center of grid cells. Although these two techniques are based on the Bayesian filter, the both offer two unique ways to generate opponent predictions. The performances of these models are compared in Chapter 6.

Chapter 6

Empirical Evaluation

This chapter presents an analysis of predictor performance in real-world scenarios. The following section introduces Counter-Strike: Source (CS:S), an online multiplayer FPS video game developed by Valve Software [48]. This game is used as a testbed to evaluate the abilities of predictors to model opponent behaviour, perform in real-time, and accurately predict opponent locations. This section also explains the game rules, common strategies, and the process used to collect CS:S game logs for testing. To compare predictors to humans at the prediction task, an online user study was held to capture the predictions for use in believability testing. Section 6.2 describes the design of the user study and provides a walkthrough of the user study. Finally, the experiment setup and evaluation is detailed in Sections 6.3 and 6.3.1 with results presented in Section 6.4.

6.1 Counter-Strike: Source

Counter-Strike: Source is a desirable testbed for a couple of reasons. First, Counter-Strike is one the most popular online FPS franchises, often achieving daily peaks of $80\,000$ people playing simultaneously [46]. The game's large player base alleviated several experiment setup issues (e.g., acquiring an ample amount of game logs containing expert gameplay) and its general popularity helped to draw participants to the user study.

Second, players engage in complex movement during CS:S games. When combat action occurs, players can be eliminated with a single shot to their avatar's head and those that manage to stay alive have no way to replenish their health. Because the survivability of a player is so volatile, typically the team with the greater number of players wins the game. Rayner shows that for both bot and human gameplay, CS:S teams that outnumber their opponents (even by one player) have a probability of winning greater than 0.5 [54]. To ensure the greatest chance of survival, expert human players resort to strategies and tactics such as utilizing cover effectively, approaching danger zones cautiously, and advancing quickly while teammates provide covering fire. Predicting the location of players executing these complex movements poses a significant challenge for both predictors and humans alike.

6.1.1 Game Rules and Objectives

One of the defining features of Counter-Strike is its goal-based gameplay.[1] Specifically, two teams are pitted against each other: the Terrorists and the Counter-Terrorists. The Terrorists are tasked with planting a bomb at one of two bomb sites (see Figure 6.1). In contrast, the role of the Counter-Terrorists is to prevent the Terrorists from setting off the bomb.

Before a game commences, each player's avatar appears in their team's start zone or *spawn area*. Players are unable to move during this intermediate phase called *freeze time*. However, they are able to purchase weapons, armour, grenades, and other miscellaneous equipment in preparation for the current game. Players must acquire what they need during this phase; no health-replenishing items are present on the game map. It is also at this time that a random Terrorist player is given the bomb.

When the game timer starts, all players are allowed to leave their spawn area and advance throughout the map. During this time, either team can win by eliminating all members of the opposing team. If the Terrorists fail to plant the bomb within a specified time limit, then the Counter-Terrorists win. However, if the bomb is been planted, the only way the Counter-Terrorists can win is by defusing the bomb before its timer goes off. In this situation, the Terrorists win if the bomb explodes or they eliminate all Counter-Terrorist players.

If a player is eliminated during the course of a game, they must wait until the next game to play again. Although waiting may appear to be an inconvenience, a typical game lasts anywhere from three to five minutes. The relatively short duration of each CS:S round allows several rounds to be played in sequence.

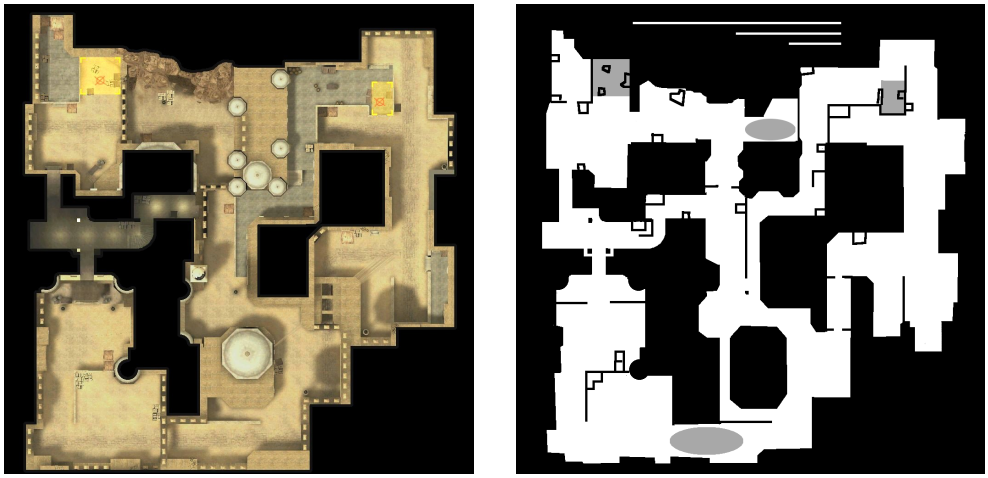


Figure 6.1: Left: Top-down view of the "de_dust2" map [48]. Right: A schematic view of the same map. Gray rectangles denote bomb sites. The Terrorist spawn area is represented by the bottom gray oval and the Counter-Terrorist spawn area by the top gray oval. The solid white lines at the top correspond to distances of 2000, 1000 and 500 coordinate units (longest to shortest).

[1] Although CS:S allows for users to create custom maps and game rules, the experiments presented herein only use the "defuse" game mode, a ruleset that is commonly played in both casual online games and professional tournaments.

6.1.2 Collecting Gameplay Data

A database of 190 CS:S game logs was collected at Fragapalooza 2006 and 2007, an annual video game competition held in Edmonton, Canada [17].[2] This venue was chosen to ensure that the game logs contained championship-level gameplay; all players gave their best efforts and no intentionally foolish behaviour was captured (e.g., players running aimlessly through the game environment).

All matches are played on the map "de_dust2", a popular choice among CS:S players for its asymmetric but fair design (see Figure 6.1). Also, each match situated five Terrorists versus five Counter-Terrorists and lasted longer than thirty seconds. The collected games were compiled into a database and then analyzed to make sure that only complete games were stored and that no players left the game during a round. Additionally, the freeze time phase was removed from the beginning of each log file.

The game logs collected serve two purposes for experiments. First, they are used to train motion models for predictors. A total of 140 logs were randomly selected to be used only for training. Second, they are used as test games for evaluating the performance of predictors. An experimental trial involves selecting a test log and then iterating over all game frames, requesting a prediction vector from a predictor at each frame. The remaining 50 game logs were used for this purpose.

6.2 The User Study

In Chapter 2, two error measures were defined for estimating the performance of predictors: Prediction Accuracy Error (PAE) and Human Similarity Error (HSE). Recall that both of these measures work by comparing a prediction vector to a "reference" vector. For PAE, the reference vector is composed of the opponents' true locations; this information can easily be found within the testing log. However, HSE compares predictors to humans and thus a source of human predictions is required to form reference vectors for this measure. The user study is designed to test human participants at the prediction task and collect the appropriate information to complete the HSE measure.

The user study consists of two major components: the *website* and the *application*. The website serves an online portal for participants to sign up for the user study, presents textual and video instructions, administers questionnaires, and provides a download link for the application. The online format was chosen because it made the user study accessible to a global audience, enabled any person to participate (as long as they had an internet connection and ran the Windows operating system), and allowed participants to complete parts of the study at their own pace. The application is a stand-alone program that presents test games to participants through a user-friendly interface. Participants are only given the same game information available to predictors must annotate the game logs with their best guesses as to where opponents are located at each point in time. Sections 6.2.1 and 6.2.2 describe the entire user study process from the perspective of a participant.

[2]Special thanks to Jeffery Grajkowski and other staff of the Alberta Ingenuity Centre for Machine Learning (AICML) who facilitated the collection of the gameplay data.

6.2.1 The Website

To participate in the user study, a participant must first create an account on the user study website. This account is used to associate the parts of the study completed by the participant (e.g., game annotations, questionnaire answers) to a unique username. In this way, the participant is able to login to the website at their leisure without losing any previous work. To preserve the anonymity of the participant, no identifying information is collected (e.g., name, email address) and any potentially-identifying information (e.g., username, password) is stored in an encrypted format. Due to this privacy policy, it is impossible to recover a participant's username or password if they happen to forget either of them. As such, the main page of the website instructs all participants to create a new account if this situation does arise.

Once an account is created, the participant is presented with a sequence of webpages called the *tutorial*. The tutorial is designed to guide a new participant through the user study process. The participant is able to navigate the sequence using "Next" and "Back" links available on each webpage. The following list details the tutorial pages in order and describes the purposes of each.

- **Introduction**: a welcome page describing the purpose of the user study and how the tutorial functions.
- **Briefing Questionnaire**: an optional questionnaire to be completed at the participant's leisure. This questionnaire requests the participant's age group, gender, and experience playing both CS:S and FPS video games in general. The full list of questions can be found in Appendix C.
- **Instructions**: a five minute instructional video on how to use the application can be viewed or downloaded. Additional information such as hotkeys and solutions to potential (but unlikely) interface issues are presented here.
- **Download**: a link to the application is provided as well as system requirements to run the program. The application was only compiled for use on computers supporting Microsoft Windows XP or Vista.
- **Intermission**: the participant is now instructed to run the application and annotate at least one game. Once complete, the participant should continue the tutorial (though there is nothing to prevent the participant from proceeding anyways).
- **Debriefing Questionnaire**: another optional questionnaire to be completed at the participant's leisure. This questionnaire allows the participant to elaborate on any ideas or tricks they used to annotate games and provides a place for the participant to report general comments about the user study. Again, the full list of questions can be found in Appendix C.
- **My Results**: a comprehensive a breakdown of the participant's annotations, ranking his performance to that of the other participants in the user study and as well as a predictor in devel-

opment. This page offers some incentive for the participant to annotate as many games as he wishes in hopes of improving their overall scores.

- **Thank You**: a complementary page congratulating the participant for completing the tutorial and encouraging the annotation of additional games.

Once the participant has reached the "Thank You" page, a menu bar appears allowing direct navigation to any tutorial webpage as well as four new ones described below.

- **My Profile**: this page provides the participant with the options to change their password and delete their account. If the participant chooses the latter option, he must confirm his decision by clicking a message prompt. All data collected from the user including questionnaire answers and game log annotations are deleted and the participant is signed out of their account.

- **Preferences**: participants can customize the game log selection mechanism of the user study application by setting filter options on the preferences page. In particular, a participant can restrict the game log selection to individual teams he wishes to annotate (the default setting allows test logs to be selected from both teams).

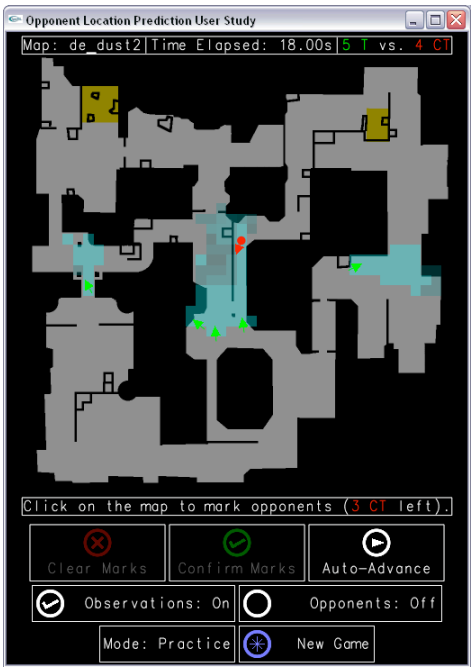
- **Support**: participants can reference this page for troubleshooting help regarding the user study website and/or application.

- **Contact**: contact information of the researchers running the user study. This is also available when creating a new user study account and thus participants do not need to be logged in to access this webpage.

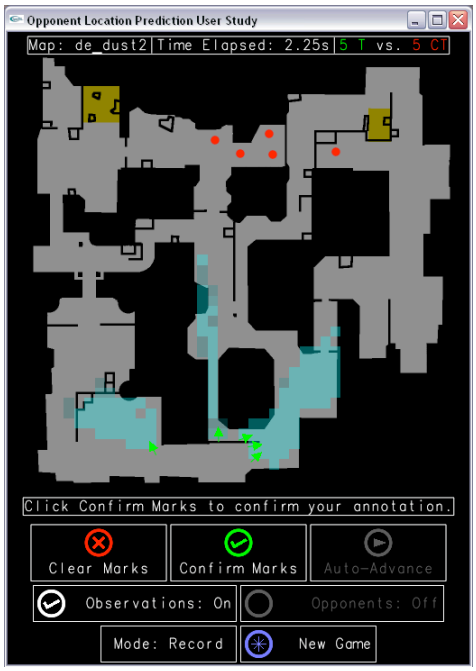
Participants may sign out of their account at any time by clicking a "Logout" link available on all webpages.

6.2.2 The Application

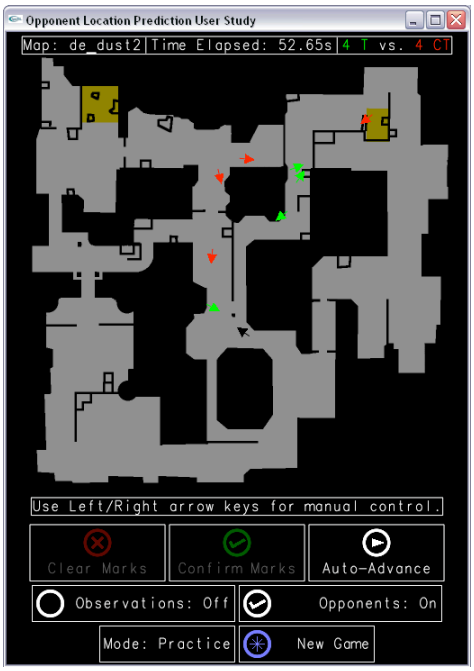
Upon running the application, the participant is presented with a window displaying a top-down view of the "de_dust2" map (Figure 6.1). A game drawn randomly from the test logs is loaded and the participant is allied with a particular team. Metadata about the current game such as time elapsed and number of players alive is displayed on a bar at the top of the window. Player locations are displayed on the map by arrows pointing in the direction which the corresponding player is facing. For example, members of the friendly team are represented by green arrows in Figure 6.2. A semi-transparent teal overlay highlights the areas of the map that the friendly players are able to see. While friendly player arrows are visible at all times, opponent arrows remain hidden from the participant's view unless observed. For example, the red arrow in Figure 6.2a is located in a friendly player's field of view and thus made visible to the participant.



(a) A game frame requiring annotation with an observed opponent. Because observed opponents are automatically marked, the participant needs to mark only the three remaining unobserved opponents.



(b) A game frame requiring confirmation. Although five marks have been placed, the participant must click the "Confirm Marks" button to continue on with the game.



(c) A game frame in practice mode with a dead opponent. The dead opponent is represented as a black arrow.

(d) A message warning the participant that he has not completely annotated the current game. This message appears when the participant clicks on either the "Mode" or "New Game" buttons.

Figure 6.2: Screenshots of the User Study application.

The participant is able to watch the game using the "Auto-Advance" button, which advances the game log frame by frame. At certain frames during the game, the participant is required to click on the map with his computer mouse, declaring his best guesses as to the current positions of unobserved opponents. Clicking places a red dot on the map, which may be cleared by clicking the "Clear" button (observed opponents are automatically "clicked" as is the case in Figure 6.2a). One click for each opponent *must* be made in order to click the "Confirm" button which allows the participant to continue with the rest of the game (Figure 6.2b). Once the end of the game is reached, the participant is able to upload their predictions to the user study website. No partially annotated games are allowed to be submitted.

Occasionally a special event may occur during a game. The participant is informed via a pop-up message if a global notice was sent to all players (e.g., the bomb was planted, the game ends). Also, if a player happens to die, the arrow representing the player is displayed in black for one frame (see Figure 6.2c), ensuring that the participant is aware of the reduction in player numbers. At any time the participant may choose to quit the program. In this situation, a dialog box is displayed warning the participant that leaving now will result in the loss of their annotations.

Several measures are taken to prevent potential biases from affecting predictions. Although participants are able to view past frames by pressing the arrow keys, they are not allowed to edit their past annotations, thus preventing the usage of future observations to improve their predictions. As well, a game cannot be annotated by the same participant more than once. After a game is submitted, it is removed from the pool of games that the participant can annotate. To ease the learning curve associated with predicting from a top-down view, participants are able to switch to a practice mode at any time by clicking the "Mode" button. The practice mode allows participants to display the locations of enemy places by toggling the "Observations" button, however they are unable to upload their annotations to the user study website. To reduce participant fatigue, the application only allows every fifth game log frame to be annotated and no prediction time limit is imposed.

6.2.3 Statistics

In total, 137 people created accounts on the user study website however only 28 participants provided annotations. Of the 28 participants, only one identified herself as a female in the briefing questionnaire. The participants annotated 98 games, 50 from the perspective of the Terrorists and 48 from the Counter-Terrorists. Unfortunately, not all humans are equally skilled at tracking moving objects. Green and Bavelier show that video game players have increased attentional capacity compared to persons who do not play video games [19]. Also, having a prior knowledge of typical CS:S gameplay is an advantage when predicting opponent positions. To compare predictors with a challenging alternative, only the annotations provided by human experts are used in the following experiments. An "expert" is defined as a participant who claims to have at least one year experience playing CS:S and 25 participants met this criteria. As such, the Human Similarity Error measure

utilizes the 38 test games in which Terrorist positions were predicted by human experts (41 games for Counter-Terrorists).

6.3 Experiment Setup

Because hidden semi-Markov models and particle filters are general frameworks, there are several configurable properties that are inherited by predictors. Table 6.1 lists all the variable parameters of a predictor and the corresponding values tested. The "Model Type" parameter indicates that a predictor uses semi-hidden Markov models or particle filters with either 500, 1000, or 2000 particles each. For example, a PF(2000) predictor tracking five opponents would update 10 000 particles on every time step. "Model Order" is discussed in detail in Section 6.4.4. The "Shared Data" parameter represents how much training data is shared among the five motion models used in each predictor (the purpose of this parameter is described in Section 6.4.3). Predictors sharing "All" of the data train each model on all 140 training logs. In contrast, a "None" predictor splits the training logs randomly into five groups and each group in turn is used to model a single opponent. Predictors sharing "Half" of the data split the training longs randomly into two groups; the first group is shared across all modes while the second group is split up in a manner similar to the "None" predictors. Finally, "Grid Cell Size" denotes the coordinate unit length and width of grid cells in G. The dimensions of these cells were chosen to keep a relatively consistent difference between the number of cells composing each grid. In total, 384 unique predictors were constructed based on these properties and their performance results are presented in Section 6.4.

Table 6.1: Experiment Parameters and Tested Values.

Parameter	Values
Opposing Team	Terrorist, Counter-Terrorist
Model Type	HSMM, PF(500), PF(1000), PF(2000)
Model Order	$1, 2$
Shared Data	None, Half, All
Grid Cell Size	$550, 450, 400, 350, 325, 300, 275, 255$

There are also several configuration parameters that are arbitrarily fixed for all experiments. The dimensions of cubes in V is set to 100 coordinate units.[3] As well, to make sure that calculations involving the duration function remain tractable, the time limit in which the system can remain in the same state (i.e., $\hat{d}$) is set to 10. Therefore, if a trajectory in a game log indicates that the player remains in a grid cell longer than 10 time steps, the player is assumed to have transitioned after the tenth step. No self-transitions were allowed between states in the transition function. Finally, the player's view cone is limited to a 90° field-of-view.

[3]For comparison, a player's in-game character stands 64 units high and can move at a maximum velocity of 320 units per second.

6.3.1 Evaluating Performance

In all experiments, the performance of a predictor is measured by the Prediction Accuracy Error (PAE) and Human Similarity Error (HSE). These two measures are reported in CS:S coordinate units and quantify the performance of a predictor from several game frames over several testing game logs. The error for a given game frame t can be represented by $F(X_t, Y_t)$ where X_t the prediction vector and Y_t is the reference vector. Specifically, Equation 6.1 is used to calculate $F(X_t, Y_t)$ where π is a permutation over elements in Y_t and n_t is the number of opponents alive. In brief, this value is the average error from a matching of predictions to reference points that minimizes the cumulative error between matched pairs. The error for a game is then calculated as the average error from all frames in the game. Finally, the PAE or HSE error reported for each predictor is the average error over all 50 testing game logs. A detailed explanation of this entire process can be found in Appendix A.

$$F(X_t, Y_t) = \min_{\pi} \frac{1}{n_t} \left(\sum_{i=1}^{n_t} D(x_t^i, y_t^{\pi(i)}) \right) \tag{6.1}$$

6.4 Experiment Results

This section presents the results of the thesis across four areas of interest. Section 6.4.1 compares and contrasts predictors based on the "Model Type" parameter mentioned in Section 6.3. Section 6.4.2 evaluates the benefit of using trained motion models on human gameplay data. Section 6.4.3 investigates the effects of sharing various amounts of data among motion models during training. Finally, the differences between using first-order and second-order models is analyzed in Section 6.4.4.

The goal of a predictor is to minimize both the PAE and HSE as much as possible. However, these measures may be *antagonistic* at times; that is, in some situations the good performance of a predictor on one measure negatively impacts its performance on the other. Given these two divergent measures of success, how can one claim that a predictor is "better" than another? One solution is to weight skillfulness and believability by some arbitrary "importance" values. However, this solution relies on the opinion of the person interpreting the experiment results. An unbiased approach is to focus on non-dominated performance. A predictor p is considered *non-dominated* if no other predictor r exists such that: 1) all errors of r are equal to or less than those of p, and 2) r has at least one error strictly less than that of p. The class of non-dominated predictors are called *Pareto-optimal* and such predictors can be considered "no worse off" than any other in the class. Figure 6.3 illustrates how both dominated and non-dominated predictors can represented by a scatter plot.

6.4.1 Model Types

In Figures 6.6 and 6.7, two classes of predictors are plotted for each model type. The "Average" class consists of a single point representing the average performance over all predictors sharing the

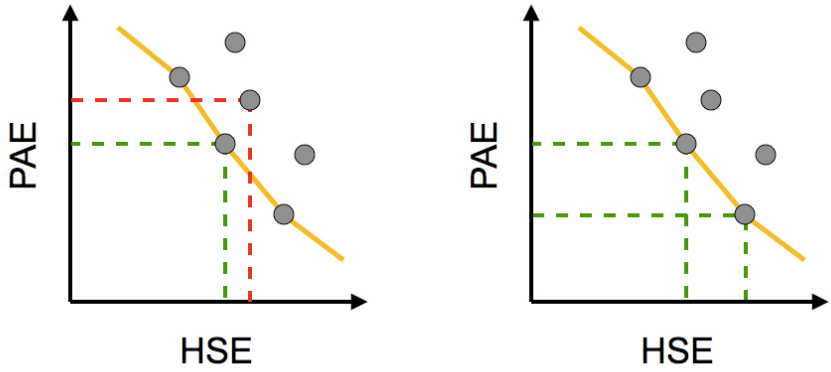


Figure 6.3: Examples of Pareto-optimal points on a scatter plot. The yellow line intersects those points that form the *Pareto frontier* (i.e., are non-dominated). Left: The point connecting the red lines is dominated by the point connecting the green lines. Right: The points connecting the green lines are both non-dominated.

same model type. The "Pareto" class consists of all non-dominated predictors and thus more than one point may be plotted for each model type. This latter class depicts the performance to be gained if one were to spend time tuning parameters to a particular video game.

To evaluate the success of predictors, two performance thresholds are marked on the graphs by dotted lines. These thresholds are drawn from the performance of human experts at the OLP task. Figure 6.4 illustrates this error, which can be viewed in two ways: as the PAE of a human's predictions and as the HSE of a perfect predictor. With regards to Figures 6.6 and 6.7, the vertical dotted lines represent the human experts' PAE while the horizontal lines represent a perfect predictor's HSE. A prediction can score both a PAE and HSE of 0 only when the prediction, the reference point specified by a human expert, and the opponent's true location are the same coordinates. However, this is often not the case and thus there is a limit on how low a predictor's average error can be. The diagonal dotted lines indicate the optimal performance achievable by any predictor; no predictor can be plotted below these lines according to the following theorem.

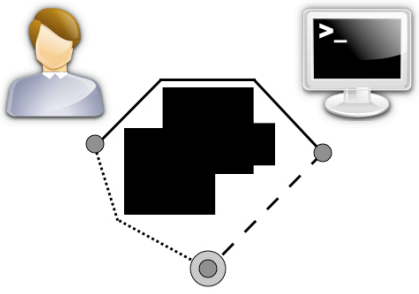


Figure 6.4: Relationship of error measures (an extended version of Figure 2.3). The dotted line denotes the error between a human's prediction and the true position of an opponent.

Theorem 6.4.1. *Let predictor performance be graphed as coordinates (x, y) on a scatter plot where the x-axis and y-axis correspond to the PAE and HSE respectively. No point will fall in the triangle formed by the x-axis, y-axis, and the line $y = -x + z$, where z is the average distance between human predictions and actual opponent locations.*

Proof. Let us rewrite the line equation from the perspective of a single predictor. Consider three vectors of Cartesian points with size n: the actual locations of opponents O, human predictions H, and the predictor's predictions P. The shortest in-game path between any pair of points (a, b) is defined by the function $D(a, b)$. Because the PAE and HSE measures represent the average distance error over several predictions, we can replace y with $\text{avg}(D(p_i, h_i))$ and x with $\text{avg}(D(p_i, o_i))$ where i indexes each vector. In a similar way, z can be replaced by $\text{avg}(D(o_i, h_i))$. Figure 6.5 illustrates how this notation applies to Figures 6.6 and 6.7.

We aim to show that $\text{avg}(D(p_i, h_i)) < -\text{avg}(D(p_i, o_i)) + \text{avg}(D(o_i, h_i))$ is impossible. The inequality can be written as $\text{avg}(D(p_i, h_i)) + \text{avg}(D(p_i, o_i)) < \text{avg}(D(o_i, h_i))$. Let us assume that a predictor performs within the region. Because $\text{avg}(x) + \text{avg}(y) = \text{avg}(x + y)$ (where x and y are vectors of the same length), we can state $\text{avg}(D(p_i, h_i) + D(p_i, o_i)) < \text{avg}(D(o_i, h_i))$. If this latter expression is true for the average, then it must also be true for (at least) one indexed set of points. We denote this set by dropping the index: $D(p, h) + D(p, o)) < D(o, h))$. If we consider each point to be a vertex of a triangle, the distances between these points form the triangle edges. However, this inequality contradicts the triangle inequality: $D(p, h) + D(p, o) \geq D(o, h)$. □

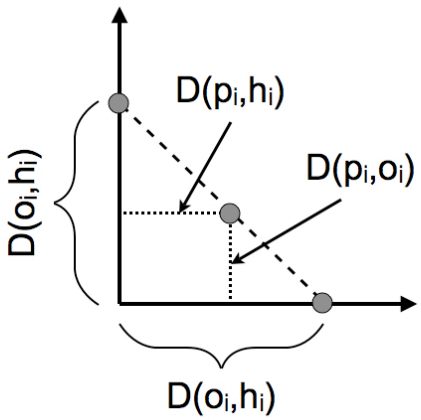


Figure 6.5: Relationship between $D(p_i, o_i)$, $D(p_i, h_i)$, and $D(o_i, h_i)$.

Predicting Terrorist Opponents

Figure 6.6 illustrates the performance of predictors tracking players on the Terrorist team. Foremost, two Pareto HSMM predictors display the best performance out of all four model types. Specifically, they make more accurate predictions than human experts and any mistakes they make are closer

Table 6.2: Terrorist Pareto Predictor Performance with Mean Update Times per Model ($\pm$ Standard Error of the Mean).

Type	Order	Share	Grid Size	Mean PAE	Mean HSE	Mean Upd. Time
HSMM	1	None	450	1142.64	1179.29	$0.55_{\pm 0.11}$ ms
HSMM	1	None	550	1160.19	1168.18	$0.26_{\pm 0.04}$ ms
PF(500)	2	None	300	1240.13	1337.57	$44.82_{\pm 0.17}$ ms
PF(500)	2	Half	275	1253.25	1313.97	$45.47_{\pm 0.19}$ ms
PF(500)	2	Half	255	1259.8	1307.14	$46.5_{\pm 0.21}$ ms
PF(500)	2	Full	275	1274.4	1304.54	$45.69_{\pm 0.21}$ ms
PF(500)	2	None	255	1285.24	1290.49	$46.62_{\pm 0.22}$ ms
PF(1000)	2	Full	325	1223.04	1299.13	$87.94_{\pm 0.26}$ ms
PF(1000)	2	None	300	1226.11	1291.58	$89.92_{\pm 0.33}$ ms
PF(1000)	1	Full	275	1236.93	1283.93	$91.5_{\pm 0.42}$ ms
PF(1000)	1	None	350	1285.74	1278.47	$86.64_{\pm 0.25}$ ms
PF(2000)	2	Full	275	1181.5	1281.47	$185.23_{\pm 0.56}$ ms
PF(2000)	1	Full	325	1185.01	1273.39	$178.59_{\pm 0.5}$ ms
PF(2000)	1	None	300	1204.9	1236.91	$181.69_{\pm 0.53}$ ms

to human predictions than a perfect predictor. An interesting observation is that these predictors use large grid cell sizes (see Table 6.2) and thus there are only a few map coordinates (namely, the center of grid cells) where HSMM predictions are possible. These coordinates are spaced out over the map, which may be an advantage in situations where an opponent location is relatively uncertain. For example, at the beginning of the game, Terrorist players have the option to travel along three divergent routes to the bomb sites. A large grid cell size forces the predictor to make general estimates which helps to minimize the PAE when an opponent could be in one of several places that are far apart.

Table 6.2 lists the configurations of each Pareto predictor plotted in Figure 6.6. The Pareto HSMM predictors have a very small update time (less than 0.6 milliseconds) which makes them suitable for implementation in a commercial video game.[4] Although the Pareto PF predictors require comparatively more time to execute an update, all motion models are highly parallel and may be made practical by offloading computations to separate threads in systems with several CPU/GPU cores.

With regards to the PF predictors, an expected trend is observed: these predictors have lower HSE and PAE values as the number of particles increases. While the majority of these predictors do not perform within the region formed by the dotted lines, the Average predictors for all model types have a PAE within 100 coordinate units of the human experts' performance. Because all model types perform well across most grid cell sizes, the direction and velocity features seem sufficient to capture Terrorist player movement effectively. Section 6.4.2 examines this particular issue in-depth.

[4] All experiments were run on a 2.8GHz Intel Core 2 processor with 2GB of RAM.

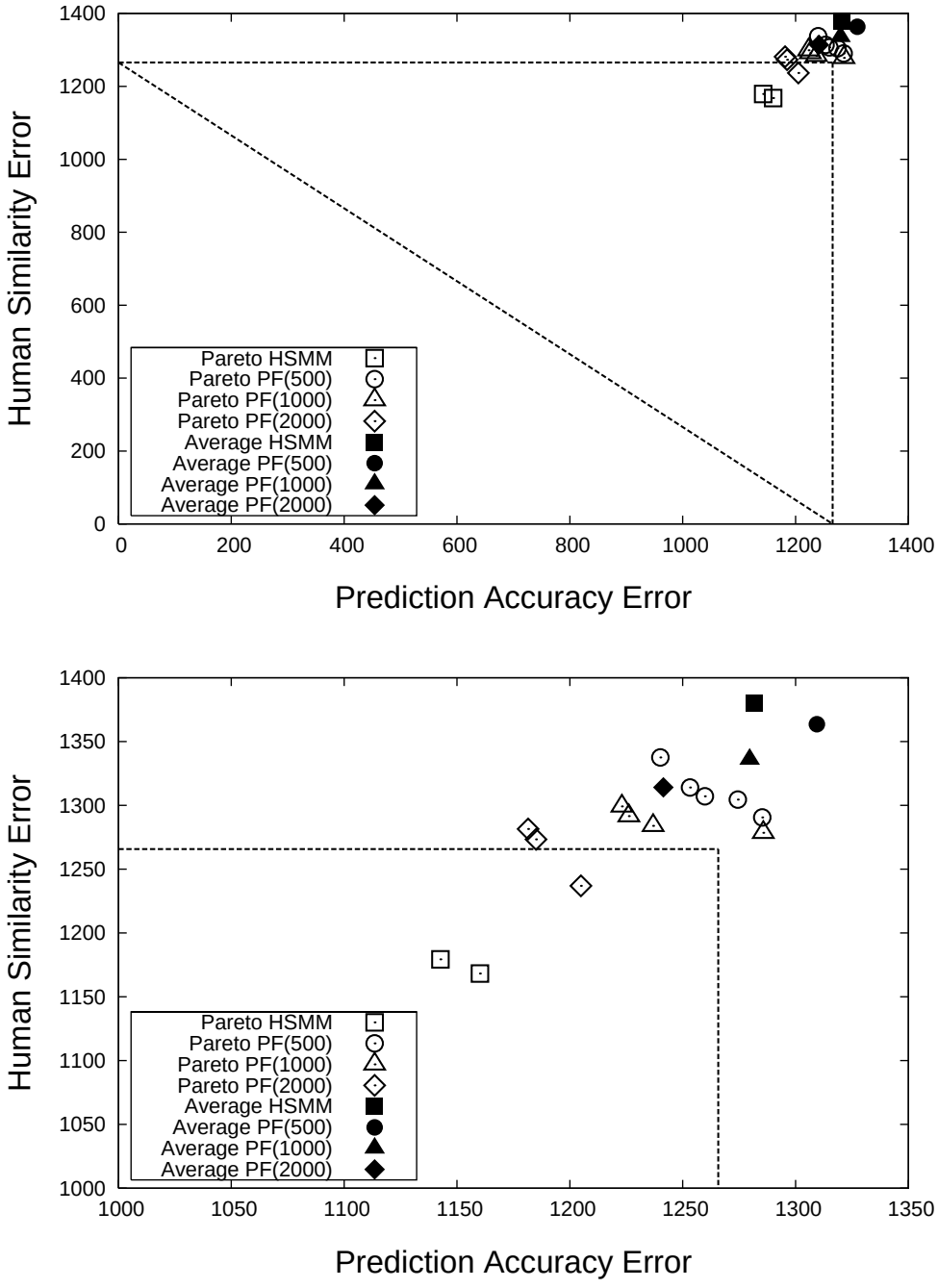


Figure 6.6: Human Similarity Error versus Prediction Accuracy Error for Terrorist Predictors by Model Type. The bottom plot is a fragment of the top plot.

Table 6.3: Counter-Terrorist Pareto Predictor Performance with Mean Update Times per Model ($\pm$ Standard Error of the Mean).

Type	Order	Share	Grid Size	Mean PAE	Mean HSE	Mean Upd. Time
HSMM	2	None	350	1147.91	1240.93	$36.1_{\pm 0.13\text{ ms}}$
HSMM	2	None	325	1155.39	1229.77	$52.85_{\pm 0.18\text{ ms}}$
PF(500)	2	None	255	1164.03	1234.54	$47.39_{\pm 0.21\text{ ms}}$
PF(500)	1	Half	255	1178.24	1222.23	$47.07_{\pm 0.15\text{ ms}}$
PF(500)	2	Half	255	1187	1216.44	$47.85_{\pm 0.23\text{ ms}}$
PF(500)	2	Full	350	1191.19	1213.88	$43.72_{\pm 0.14\text{ ms}}$
PF(500)	2	Half	350	1195.76	1192.88	$43.66_{\pm 0.12\text{ ms}}$
PF(500)	2	Full	300	1208.56	1186.42	$45.68_{\pm 0.18\text{ ms}}$
PF(1000)	2	Full	350	1154.68	1205.75	$87.89_{\pm 0.2\text{ ms}}$
PF(1000)	2	None	300	1160.69	1184.85	$91.04_{\pm 0.23\text{ ms}}$
PF(1000)	1	None	350	1170.93	1177.84	$88.87_{\pm 0.36\text{ ms}}$
PF(1000)	1	Half	450	1200.81	1159.39	$85.99_{\pm 0.21\text{ ms}}$
PF(2000)	2	Full	275	1142.25	1181.77	$187.05_{\pm 0.41\text{ ms}}$
PF(2000)	2	Half	275	1156.99	1179.42	$190.92_{\pm 0.43\text{ ms}}$
PF(2000)	2	Half	350	1162.35	1166.41	$178.45_{\pm 0.37\text{ ms}}$
PF(2000)	1	Full	350	1164.43	1165.69	$178.17_{\pm 0.36\text{ ms}}$
PF(2000)	1	Half	350	1170.94	1160.98	$178.26_{\pm 0.35\text{ ms}}$

Predicting Counter-Terrorist Opponents

Figure 6.7 presents the results of predictors tracking Counter-Terrorist opponents. When comparing absolute error values, the Pareto Counter-Terrorist predictors perform better than their counterpart Terrorist predictors in Figure 6.6. Unfortunately, the relative error between the predictors and performance thresholds is large with no predictor crossing any threshold. This result may be attributed to a particularly low PAE achieved by human experts. A likely explanation is that humans comprehend the intricacies of Counter-Terrorist movements. While Terrorist strategies focus on quickly traversing the map to get to the bomb sites, Counter-Terrorists act defensively, relying on ambushing and standoff tactics that require players to remain stationary for extended periods of time. The imposed limit of $\hat{d} = 10$ also contributes to poor predictor performance because it prevents the motion models from accurately capturing such tactics. To accurately model Counter-Terrorist behaviour, a value for $\hat{d}$ greater than 10 and features other than trajectory direction and velocity may need to be considered when training predictors.

Predicting with Gaussian Noise

As discussed in Section 1.3, cheating can be used to improve the PAE of a predictor. However, a bot that knows the locations of its opponents with an uncanny consistency may not be considered human-like. This section investigates the question whether masking cheating by random noise can result in more believable predictions. Figures 6.8 and 6.9 plot several cheating predictors by asterisks. A perfect predictor is located on the y-axis (because its PAE is always 0) and estimations by

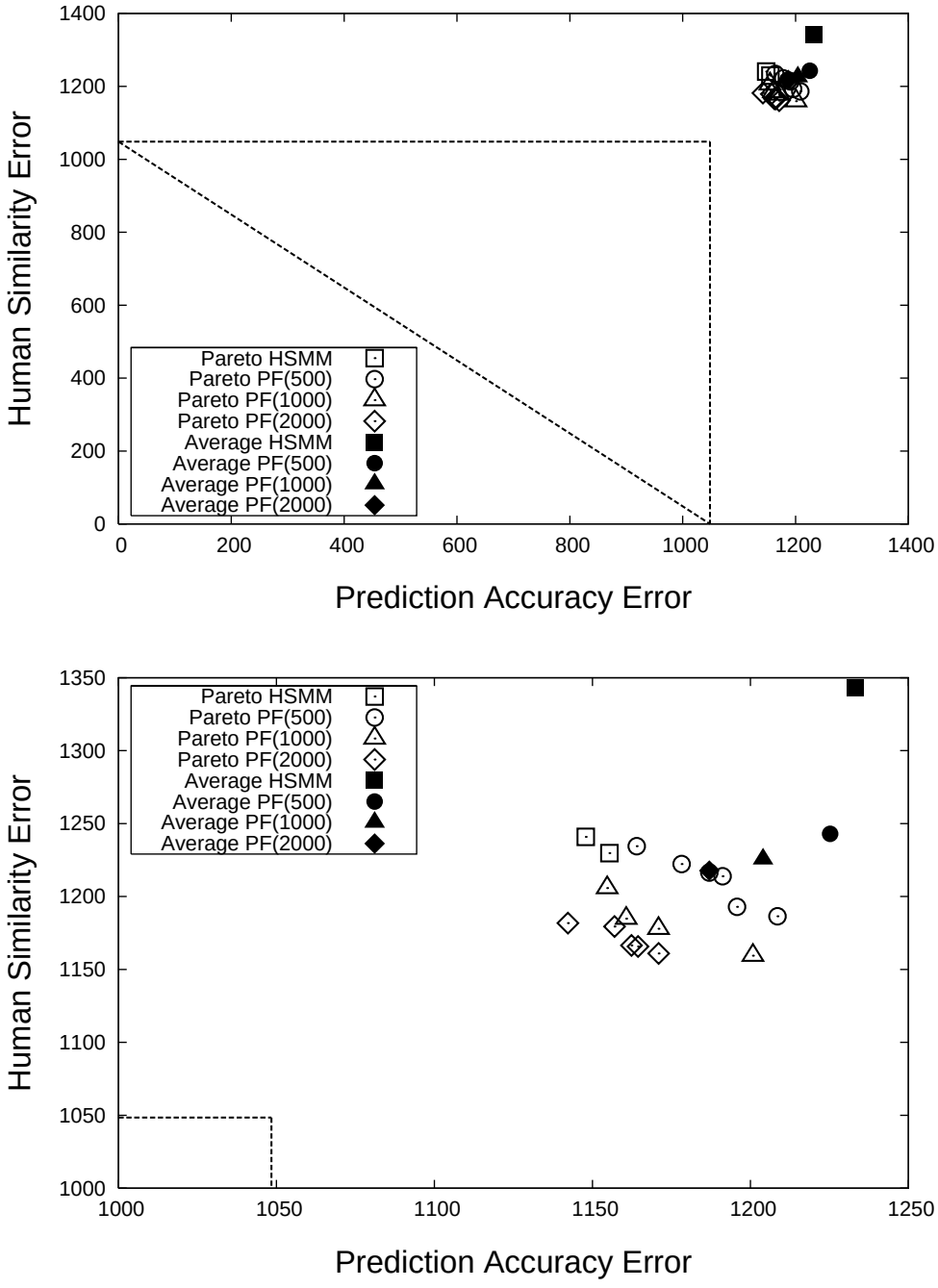


Figure 6.7: Human Similarity Error versus Prediction Accuracy Error for Counter-Terrorist Predictors by Model Type. The bottom plot is a fragment of the top plot.

this predictor are perturbed by varying amounts of Gaussian noise, resulting in the performances represented by the other asterisks. The set of Pareto predictors from Figures 6.6 and 6.7 are also plotted for comparison. Finally, a least squares regression line is fit to the set of cheating predictors to emphasize any performance trend.

In both Figures 6.8 and 6.9, an increase in the standard deviation of Gaussian noise is met with a corresponding increase an asterisk's PAE value. This is clear by observing the steady incline of the regression lines. As the PAE of the noisy predictors approach that of human experts (specifically, the predictor with standard deviation of 1500), the HSE is significantly larger than that of all non-cheating predictors. In this way, the mistakes made by the HSMM and PF predictors can be considered more human-like than those made when using Gaussian noise to hide cheating.

Gaussian noise obfuscation has a greater HSE than trained predictors because map topology is not taken into account. While a prediction perturbed by noise may be close to the true opponent location in Euclidean space, the shortest path between these points in the game environment may be long if there happens to be a wall or obstacles in the way. Conversely, human experts understand how the environment layout can restrict an opponent's movement and they use this information to inform their predictions. An alternative way to mask cheating could use a random walk or other in-game path generator to perturb the opponent's position. Though such a technique resolves the topology issue, this idea is not investigated in this work (though it would be worthy of further research). In summary, human experts do make mistakes but their mistakes are not normally distributed.

6.4.2 Training Motion Models

The ability of predictors to model opponent movements has thus far been presented as an advantage over other competing prediction methods. However, because of the complex motions exhibited in CS:S gameplay, it is of interest to know whether or not the training can capture any useful knowledge that would improve predictor performance. To investigate this issue, Figures 6.10 and 6.11 represent each of the 384 predictors as a single point. For each predictor, a second one is tested using a uniform motion model; that is, the model is encoded such that an opponent in some grid cell g has an equal probability to transition to any traversable cell adjacent to g. For this test it is expected that the predictors with uniform motion models should predict worse than the trained predictors.

Figure 6.10 shows almost all trained predictors performing with lower errors than the majority of uniform predictors. This can be attributed to the typical attacking strategies for the Terrorist team. Because the goal of the Terrorists is to plant the bomb at the opposite end of the map from their spawn area, the general positions of Terrorists will progress to the other end of the map over the course of the game. Figure 6.10 supports the proposition that this Terrorist movement is captured in the motion models. Conversely, the Counter-Terrorist case does not show as drastic a difference between trained and uniform models (Figure 6.11). Instead, most trained predictors perform only as well as the best uniform models. Considering that human experts were able to predict with a

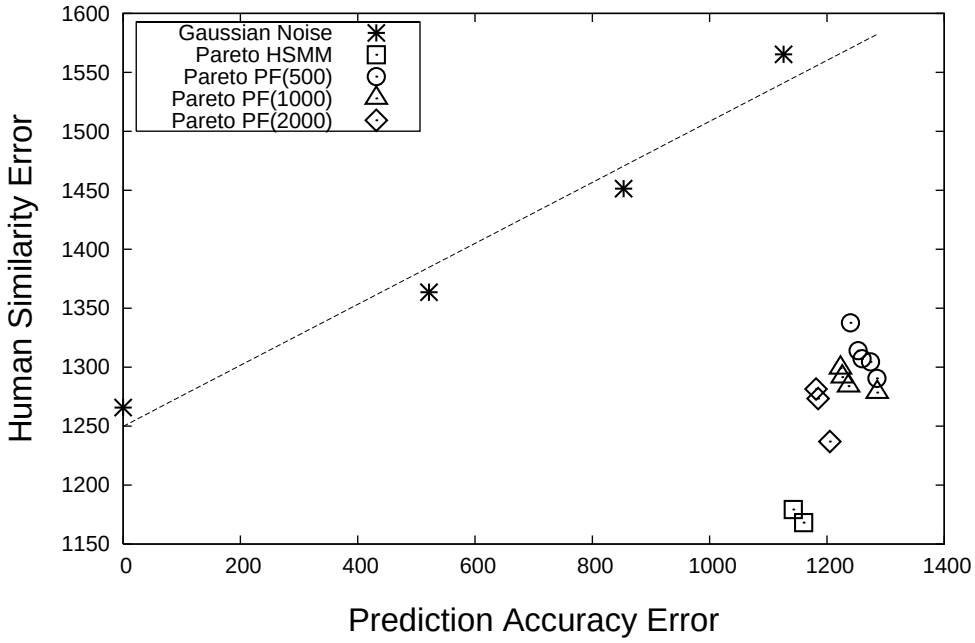


Figure 6.8: Human Similarity Error versus Prediction Accuracy Error for Terrorist Predictors. The asterisk on the y-axis represents a perfect predictor while other asterisks are perturbed by Gaussian noise using standard deviations of 500, 1000, and 1500 coordinate units (left to right). The dotted line is a least squares linear regression indicating the performance trend of noisy predictors.

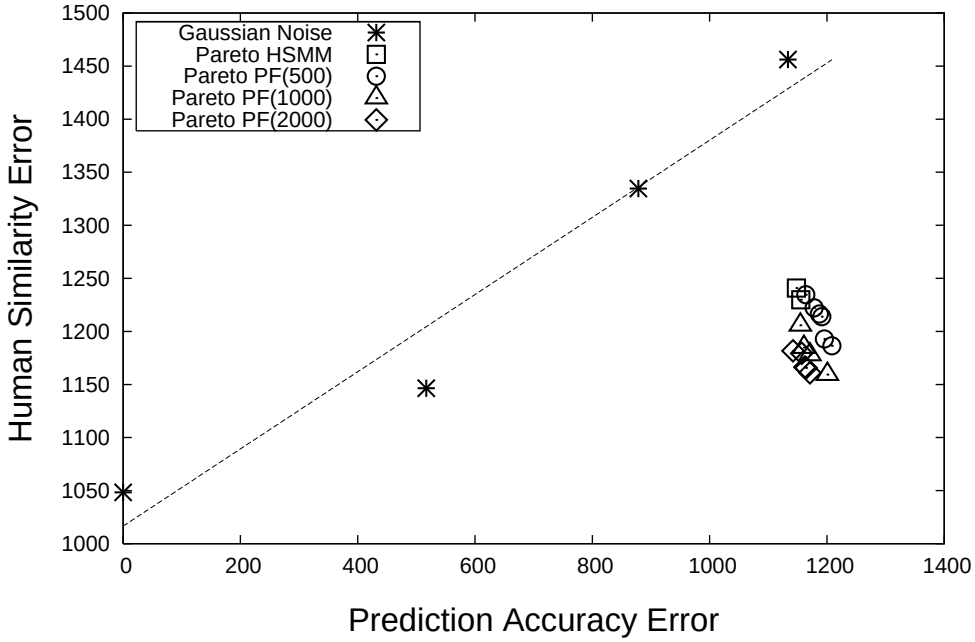


Figure 6.9: Human Similarity Error versus Prediction Accuracy Error for Counter-Terrorist Predictors. The asterisk on the y-axis represents a perfect predictor while other asterisks are perturbed by Gaussian noise using standard deviations of 500, 1000, and 1500 coordinate units (left to right). The dotted line is a least squares linear regression indicating the performance trend of noisy predictors.

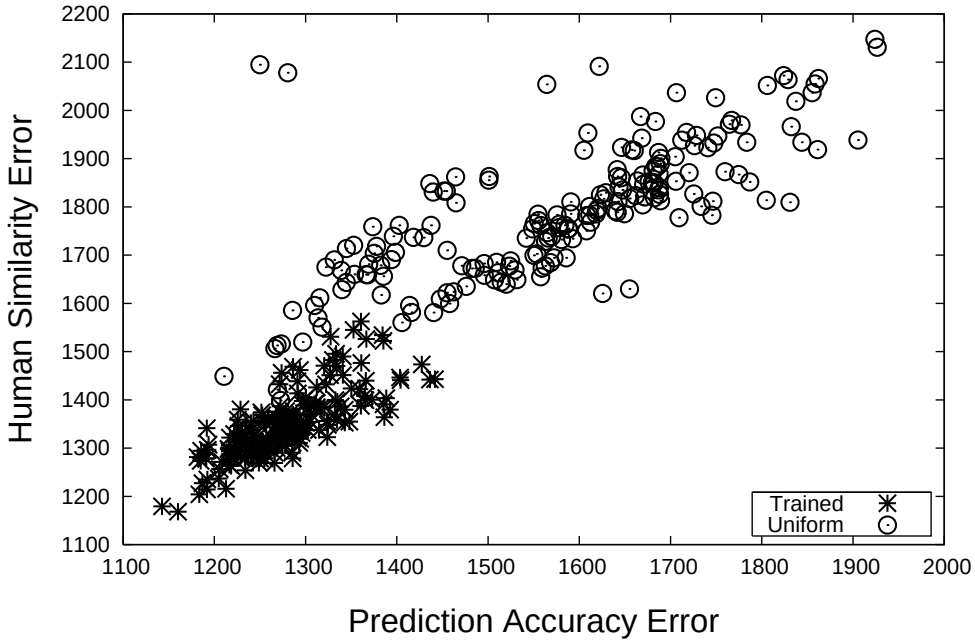


Figure 6.10: Human Similarity Error vs. Prediction Accuracy Error for Terrorist Predictors.

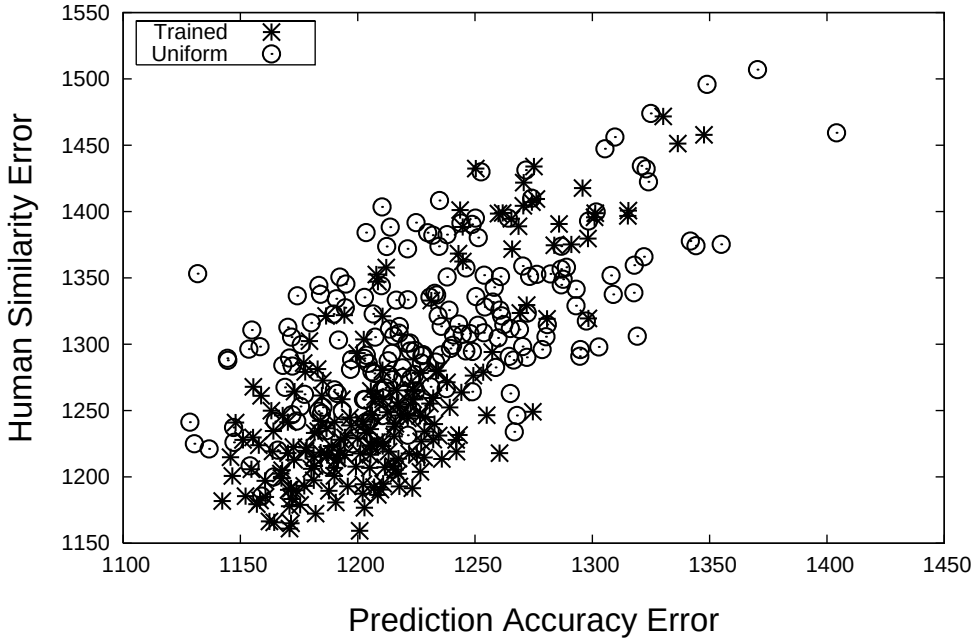


Figure 6.11: Human Similarity Error vs. Prediction Accuracy Error for Counter-Terrorist Predictors.

low PAE in Section 6.4.1, it is likely that both the trained and uniform predictors are both poor at predicting the positions of Counter-Terrorist opponents.

6.4.3 Sharing Training Data

It was shown in Section 6.4.2 that predictors trained on Terrorist data perform better than uniform predictors. An important question to ask is how much data is required to produce these successful motion models. Are all 140 training logs necessary to learn the transition and duration models or can it be done with fewer? Figures 6.12 and 6.13 present a comparison between three types of predictors sharing training data as explained in Section 6.3.

PF model types do not show any significant difference by varying the amount of training data per model (these figures can be found in Appendix B). However, the HSMM predictors that do share logs among their models typically perform poorly compared to those that do not. One explanation for this phenomenon can be drawn from games were few opponent sightings are made. Consider the start of such a game where an HSMM predictor initializes its probability distributions to the prior. Note that at this point all distributions are equal and each model will update its distribution in exactly the same way until an opponent is sighted. The result is that all predictions returned from the models will be a single point, representing the strategy of all friendly players grouping together wherever they go on the map. While the trajectory data may be accurate in defining player movement, predicting every opponent to follow this strategy results in a high PAE. Note that particle filters do not suffer from this issue because they randomly select a particle when predicting each opponent. The "All" predictor variants, while being less informed (each model trained on 28 logs), offer varied motion models that resolve the above issue for games that contain very few sightings. Therefore, it can be concluded that very few game logs are required to identify the common trajectories of players and that players disperse more often than group together when no opponents have been sighted.

6.4.4 Motion Model Orders

Thus far, the motion models presented in Chapter 5 and tested the previous experiments are "first-order" models; that is, the probability of an opponent moving to some grid cell g_{t+1} at time $t+1$ is conditional only upon the probability of the opponent occupying the current grid cell g_t and the number of times steps until the next transition. Note that this formulation only allows for an opponent's future position to be dependent upon his past position, not his past *trajectory*. To elaborate on this distinction, consider a horizontal hallway where typical gameplay involves players moving down the hallway in a straight line (no backtracking). Also, assume that the frequency of observing players starting from the left and moving to the right is the same as those moving from right to left. If an opponent is suspected to be in the middle of the hallway, a first-order model would predict with equal probability that the opponent would move to the left or right, even if the opponent had been previously sighted moving in a particular direction. In contrast, the transitional probabilities of a

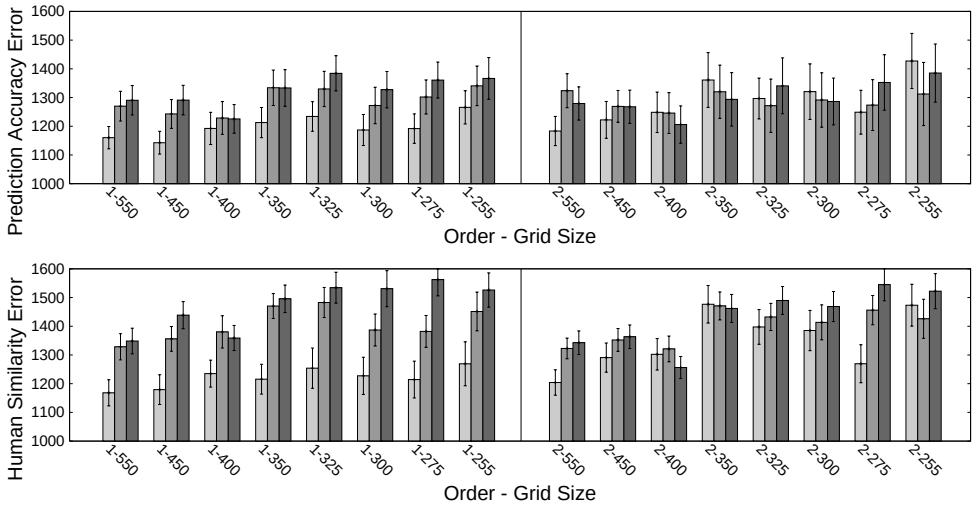


Figure 6.12: Shared Training Data of HSMM Terrorist Predictors. From left to right (i.e., light to dark), the bars represent the performance of predictors sharing "None", "Half", and "All" of the training data among the motion models.

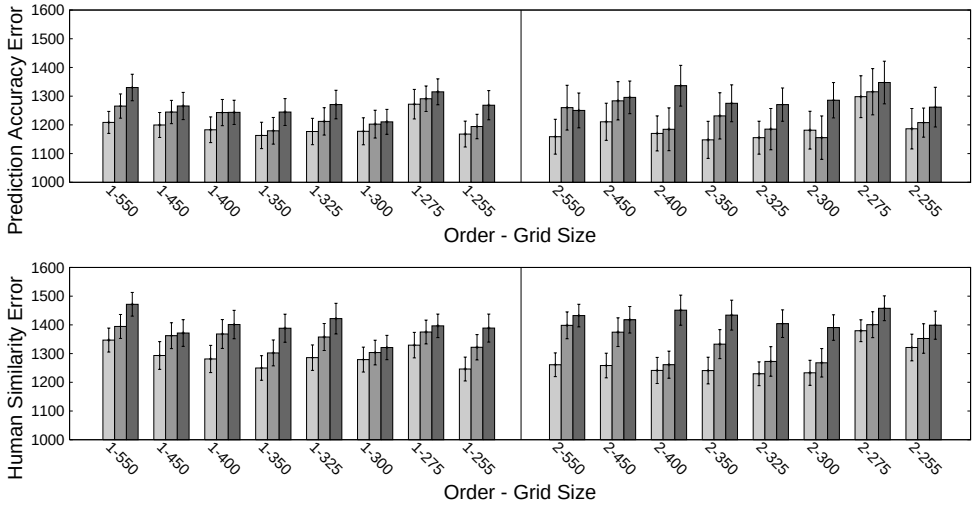


Figure 6.13: Shared Training Data of HSMM Counter-Terrorist Predictors. From left to right (i.e., light to dark), the bars represent the performance of predictors sharing "None", "Half", and "All" of the training data among the motion models.

"second-order" model are conditioned on the previous *two* cells occupied by the opponent. This formulation resolves the above prediction issue by using the past direction of the opponent (as opposed to the location only) to predict the current direction in which he is heading.

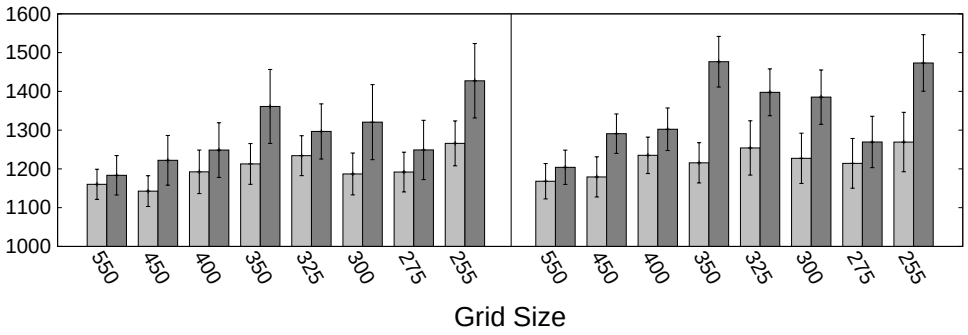


Figure 6.14: First and Second-Order HSMM Terrorist Predictors. The left image plots Prediction Accuracy Error along the y-axis while the right image plots Human Similarity Error. The light bars represent first-order predictors and the dark bars represent second-order.

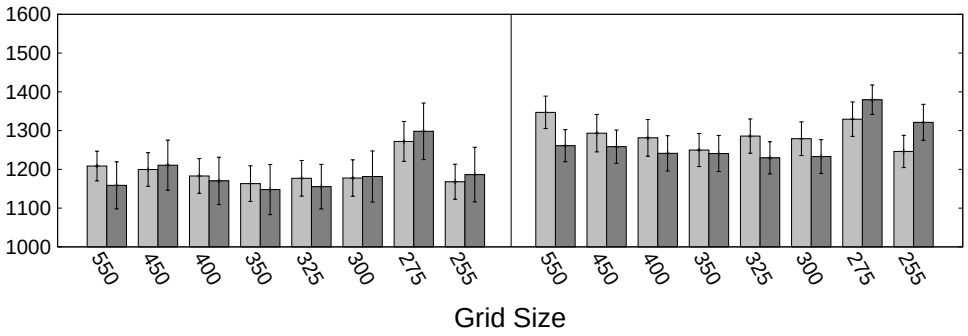


Figure 6.15: First and Second-Order HSMM Counter-Terrorist Predictors. The left image plots Prediction Accuracy Error along the y-axis while the right image plots Human Similarity Error. The light bars represent first-order predictors and the dark bars represent second-order.

Figure 6.14 shows that first-order models perform more adequately than second-order models for HSMM predictors estimating Terrorist positions. The same cannot be said for Counter-Terrorist opponents (Figure 6.15) where larger grid cell sizes show a marginal but insignificant improvement for second-order models. Also, in parallel to the results of Section 6.4.3, there is no discernible difference between first and second-order models for particle filters (complete results are listed in Appendix B). For second-order models to have an effect on performance, it appears that more training data is needed to properly specify the transition and duration functions. Section 6.4.2 concluded that CS:S player movement can be adequately captured by these functions and Figure 6.14 suggests that first-order models are capable of doing so.

6.5 Summary

This chapter presented the performance results of predictors evaluated according to the PAE and HSE measures. It was shown in Section 6.4.1 that predictors estimating Terrorist positions performed as well (if not better) than the human experts against the PAE measure. In addition, the best HSMM predictors perform with a lower HSE than perfect predictions. The computation time required to update these predictors is less than a millisecond, making them suitable for implementation in commercial FPS games. Moreover, both Terrorist and Counter-Terrorist predictors maintain an HSE consistent with perfect predictors. Gaussian noise is insufficient to mask cheating, instead displaying non-human errors (i.e., high HSE performance) when compared to HSMM and PF predictors at a similar PAE.

Section 6.4.2 illustrated that training predictors on expert human gameplay does improve performance along both the PAE and HSE measures. Section 6.4.3 noted that HSMM predictors perform well with varied trajectories for individual predictions and do not benefit from sharing movement data among motion models. Finally, second-order models do not show any conclusive improvement over first-order models.

Chapter 7

Discussion

Having presented the results of this work in Chapter 6, Section 7.1 of this chapter identifies some of the challenges and limitations encountered during the user study and experiments. With these issues in mind, Section 7.2 proposes a few directions in which this research can be extended. Section 7.3 concludes by discussing some novel applications of predictors as components of a bot and as non-cheating aids for human players.

7.1 Challenges and Limitations

The foremost limitation of this work is the lack of a large game log corpus. A significant collection of game logs would reduce overfitting when training on grids with very small cell sizes, thus capturing low-level movement that is not currently possible. Unfortunately, it is difficult to collect logs that are guaranteed to have competent gameplay. Ideally, a plugin should be developed that would be distributed to mass amounts of players. The plugin would record games and store them in a central research database for future analysis. A large portion of these games would involve poor or foolish gameplay and thus should be pruned from the database. This task would have to be automated, filtering undesirable gamelogs according to some criteria. However, developing a definition for "expert gameplay" is a problem in itself. The lack of data also applies to the number of participants in the user study. The study was unable to collect more than one human-annotated game for each testing log and thus the reference predictions for each game frame were generated by a single participant.

Despite all the precautions taken in the user study to ensure that participants were tested in the same manner as predictors, there were two differences that may have impacted HSE performance in favor of the participants. First, because all players are displayed as arrows in the user study application, participants were informed of the directions that both friendly and opponent players were facing. Second, participants could use friendly behaviour to inform their predictions. For example, if a friendly player was observed to be jostling back and forth with no opponent in sight, a human expert might recognize the player to be engaged in a firefight. If the opponent happened to be sighted on an *off-frame* (the brief time between frame recordings), friendly players may often hide

behind cover, reducing the chance that the opponent is sighted again. In this scenario, a predictor would have no knowledge of the sighting while a participant might be able to deduce the opponent's location through indirect means.

7.2 Future Work

There are several avenues for extending the work presented in this thesis. Predictors currently learn only the movement directions and velocities of players from gameplay data. Given an ample corpus of game logs, learning can be conditioned on additional known game state features. For example, if it is assumed that an opponent's weapon can be determined when sighted, then associating weapons with observed movement may improve the quality of information captured by the transition and duration functions. In this way, the movement policy of a sniper can be maintained separately from that of a player carrying a shotgun. As well, an increased limit on the number of time steps an opponent is allowed to remain in a single state (i.e., $\hat{d}$) should also be explored, though doing so will impact the time to compute updates.

Section 5.1 states that factored models were chosen in favor of joint-position models because of the intractable update calculations involved with the latter. The downside to this decision is that predictors are unable to learn any synchronized player movements, an important type of coordination and teamwork. Future work involving this issue might focus on developing a hybrid system where joint models are trained for subsets of a team (e.g., for pairs of players).

One drawback to the point predictions described in Section 2.1 is that uncertainty cannot be accurately encoded in the representation. For instance, a situation may arise in-game where it is equally probable for an opponent to exist in two distinct locations yet a single point prediction must be provided. While this representation was avoided in this work because it would complicate the process of specifying a prediction, probability distributions could be used in place of point predictions to capture uncertainty in a prediction. For instance, two distributions could be compared to each other using a Mahalanobis distance metric to account for the shortest in-game paths between points in the environment [34].

7.3 Applications

The original purpose of a predictor was to serve as a stand-alone module that can be inserted into the decision-making system of a bot. The module would impart the bot with human-like reasoning about opponent locations in real-time. Although the experiments in this work used Counter-Strike: Source as a testbed, efforts were made to ensure that predictors could be applied to any game where spatial navigation is important. However, beyond this purpose there are several novel applications in which predictors can play an integral role.

By itself, a predictor is strictly informative; its sole task is to provide an estimation of opponent

locations. However, predictions can obviously be used to inform a bot's controlling mechanisms. Consider the orientation of a bot's field-of-view. A realistic view policy would be to look in the direction where the greatest threat is expected to originate. Predictions could be used as simple representation of threat. However, if a bot's view to a prediction is obstructed, the bot should not display unintelligent behaviour (e.g., staring at walls or other inanimate objects). Instead, an opponent's trajectory could be computed and the bot directed to look towards the furthest visible edge of the trajectory.

An even more interesting challenge is to make use of opponent movement projected into the future. All predictions have thus far been generated on the current time step t yet Bayesian filters have the ability to estimate $P(S_{t+n}|O_{1:t})$ where $n \geq 1$. This distribution gives an approximation of where the opponents will be after n time steps. Although the distribution degrades in accuracy as n increases, this technique can nonetheless be used to choose between complex strategies that take many time steps to execute.

Because of its non-cheating design, a predictor can be used to aid novice players in predicting opponent locations during a live game. An overlay could be constructed similar to the images in Figure 5.3 that would appear in games that enable players to view a map of the game environment. Depending on the game, the predictor might not be able to utilize all observations of friendly teammates to inform the predictor however it would be nearly undetectable because it does not need to interact directly with a game server.

Finally, one major advantage to incorporating opponent modeling into a predictor is the option to customize the motion models for teaching purposes. For example, if team A knows that it will be facing team B in an important tournament, then team A could collect logs of team B's past gameplay, learn a predictor from the logs, and then use the predictor as an overlay for training prior to the real match. This work provides predictors with this type of customizability, a small step towards a bot that can assess an opponent's tactical strengths and weaknesses by analyzing past gameplay.

Chapter 8

Conclusion

This thesis investigated the problem of predicting opponent positions in First-Person Shooter video games. First, the notions of skillfulness and believability were formalized as two performance measures. Several AI challenges that are present in modern video games were described and cheating techniques were argued as inadequate solutions to these challenges because they increased skillfulness at the expense of believability. This work proposed that AI can be both skillful and believable without cheating, and focused on the prediction problem to show that this is possible. Hidden semi-Markov models (HSMM) and particle filters (PF) were configured as *predictors*, stand-alone non-cheating modules whose purpose is to predict opponent positions in the first-person shooter (FPS) game Counter-Strike: Source. To simulate the reasoning faculties that humans use when making predictions, these predictors were trained on logs of expert human gameplay and they utilized the sensory information available to players to produce informed predictions. Predictions were graded according to the two performance measures, specifically the prediction accuracy error and the human similarity error. The latter measure was constructed using predictions provided by humans through a user study designed to test participants at the prediction problem. Results show that HSMM predictors can be as accurate as expert human players and can perform efficiently in real-time scenarios. As well, the mistakes they do make are more human-like that those made by perturbing true opponent locations with a corresponding amount of Gaussian noise. These results indicate that complex human movement in FPS games can be learned successfully from gameplay data.

Bibliography

[1] Activision. Call of Duty 4 official website. http://www.callofduty.com/CoDMW, 2009.

[2] R. Allen, P. Mcgeorge, D. Pearson, and A. B. Milne. Attention and expertise in multiple target tracking. *Applied Cognitive Psychology*, 18(3):337–347, 2004.

[3] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[4] N. Bard and M. Bowling. Particle filtering for dynamic agent modeling in simplified poker. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*, pages 515–521, 2007.

[5] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thurun. Learning motion patterns of people for compliant motion. *International Journal of Robotics Research*, 2004.

[6] C. Bererton. State estimation for game AI using particle filters. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, Pittsburgh, Pennsylvania, 2004. AAAI Press.

[7] S. Blyth and H. J. Hamilton. Crowdmixer: Multiple agent types in situation-based crowd simulations. In *Proceedings of the Second Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 15–20, 2006.

[8] M. Booth. The official Counter-Strike bot. Presentation at Game Developers Conference, 2004.

[9] D. A. Borovies. Particle filter based tracking in a detection sparse discrete event simulation environment. Master's thesis, Naval Postgraduate School, Monterey, California, March 2007.

[10] A. Bruce and G. Gordon. Better motion prediction for people-tracking. In *Proceedings of the International Conference on Robotics and Automation*, 2004.

[11] M. Buro. The othello match of the year: Takeshi Murakami vs. Logistello. *International Computer Chess Association Journal*, 20(3):189–193, 1997.

[12] M. Buro. Real-time strategy games: A new AI research challenge. In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, pages 1534–1535, Acapulco, Mexico, 2003.

[13] A. J. Champandard. How to help players notice the "I" in AI? http://aigamedev.com/discussion/notice-intelligence-bioshock, February 2008.

[14] C. H. Chen, C. Cheng, D. Page, A. Koschan, and M. Abidi. A moving object tracked by a mobile robot with real-time obstacles avoidance capacity. In *Proceedings of the Eighteenth International Conference on Pattern Recognition*, volume 3, pages 1091 – 1094, 2006.

[15] C. Darken and B. G. Anderegg. *Particle Filters and Simulacra for More Realistic Opponent Tracking*, chapter 4.6, pages 419–427. In Rabin [53], 2008.

[16] D. Doherty and C. O'Riordan. *Toward More Humanlike NPCs for First-/Third-Person Shooter Games*, chapter 5.6, pages 499–511. In Rabin [53], 2008.

[17] Fragapalooza. Fragapalooza official website. http://www.fragapalooza.com/, 2008.

[18] B. Gorman, C. Thurau, C. Bauckhage, and M. Humphrys. Believability testing and Bayesian imitation in interactive computer games. In *Proceedings of the Ninth International Conference on the Simulation of Adaptive Behavior*, pages 655–666, September 2006.

[19] C. S. Green and D. Bavelier. Action video game modifies visual selective attention. *Nature*, 423:534–537, 2003.

[20] N. Hoobler, G. Humphreys, and M. Agrawala. Visualizing competitive behaviors in multi-user virtual environments. *IEEE Visualization*, pages 163– 170, 2004.

[21] D. Isla. The virtual hippocampus: Spatial common sense for synthetic creatures. Master's thesis, Massachuettes Institute of Technology, August 2001.

[22] D. Isla. *Probabilistic Target Tracking and Search Using Occupancy Maps*, pages 379–387. In Rabin [52], 2006.

[23] R. E. Kalman. A new approach to linear prediction and filtering problems. *Journal of Basic Engineering*, 1960.

[24] M. Klaas, T. Southey, and W. Cheung. Particle-based communication among game agents. In *Proceedings of the First Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005.

[25] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[26] J. E. Laird. It knows what you're going to do: Adding anticipation to a Quakebot. In *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 385–392, Montreal, Quebec, Canada, 2001.

[27] J. E. Laird and J. C. Duchi. Creating human-like synthetic characters with multiple skill levels: A case study using the Soar Quakebot. In *Proceedings of the AAAI 2000 Fall Symposium Series: Simulating Human Agents*. AAAI Press, November 2000.

[28] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(3):1–64, 1987.

[29] J. E. Laird and M. van Lent. Human-level AI's killer application: Interactive computer games. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 1171–1178, 2000.

[30] R. A. Lastra, P. A. Vallejos, and J. Ruiz-del Solar. Self-localization and ball tracking for the Robocup 4-legged league. In *Proceedings of the Second IEEE Latin American Robotics Symposium*, September 2005.

[31] D. Livingstone. Turing's test and believable AI in games. *Computers in Entertainment*, 4(1), January 2006.

[32] H. G. Loebner. Loebner prize. http://www.loebner.net/Prizef/loebner-prize.html, 2007.

[33] B. Mac Namee. *Proactive persistent agents: Using situational intelligence to create support characters in character-centric computer games*. PhD thesis, Department of Computer Science, University of Dublin, Dublin, Ireland, 2004.

[34] P. C. Mahalanobis. On the generalized distance in statistics. In *National Institute of Science (India)*, volume 12, pages 49–55, Calcutta, India, 1936.

[35] 2K Boston. Bioshock official website, 2007.

[36] Blizzard Entertainment. Warcraft 3 official website. http://www.blizzard.com/us/war3/, 2009.

[37] Bungie, LLC. Halo 3 heatmaps. http://www.bungie.net/Online/Heatmaps.aspx, 2009.

[38] Coalescent Technologies Corporation. MTVC: Mobile virtual training capability, 2008.

[39] Gamespot.com. Call of Duty 4 scoreboard screenshot, 2009.

[40] Gridiron Technologies. Pro simulator official website, 2008.

[41] id Software. Quake II official website, 2008.

[42] id Software. Wolfenstein 3D official website, 2008.

[43] International Cyber Marketing, Inc. World Cyber Games official website, October 2008.

[44] Take-Two Interactive. 2K Bot Prize. http://botprize.org/, 2009.

[45] Valve Software. Counter-Strike: Source. http://store.steampowered.com/app/240/, 2008.

[46] Valve Software. Steam and game stats. http://store.steampowered.com/stats/, 2008.

[47] Valve Software. Team Fortress 2 statistics. http://www.steampowered.com/status/tf2/tf2_stats.php, April 2008.

[48] Valve Software. Valve games, 2008.

[49] K. Murphy. Hidden semi-Markov models (HSMMs). Technical report, University of California at Berkley, 2002.

[50] M. Newborn. *Kasparov versus Deep Blue: Computer chess comes of age*. Springer-Verlag New York, Inc., New York, NY, 1996.

[51] Z. Pylyshyn and R. Storm. Tracking multiple independent targets: Evidence for a parallel tracking mechanism. *Spatial Vision*, 3(3):1–19, 1988.

[52] S. Rabin, editor. *AI Game Programming Wisdom 3*. Charles River Media, 2006.

[53] S. Rabin, editor. *AI Game Programming Wisdom 4*. Charles River Media, 2008.

[54] D. C. Rayner. Analysing openings in tactical simulations. Master's thesis, Department of Computing Science, University of Alberta, 2008.

[55] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson US Imports & PHIPEs, second edition, November 2002.

[56] J. Schaeffer, N. Burch, Y. Bjornsson, A. Kishi-moto, M. Muller, R. Lake, P. Lu, and S. Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, July 2007.

[57] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, and D. Billings. Bayes' bluff: Opponent modeling in poker. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 550–558, 2005.

[58] F. Southey, W. Loh, and D. Wilkinson. Inferring complex agent motions from partial trajectory observations. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 2631–2637, Hyderabad, India, 2007.

[59] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[60] D. Thue. Player-informed interactive storytelling. Master's thesis, Department of Computing Science, University of Alberta, 2007.

[61] D. Thue, V. Bulitko, M. Spetch, and E. Wasylishen. Interactive storytelling: A player modelling approach. In *Proceedings of the Third Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 43–48, 2007.

[62] A. Turing. Computing machinery and intelligence. *Mind*, 59(236):443–460, 1950.

[63] F. A. J. Verstraten, P. Cavanagh, and A. T. Labiancab. Limits of attentive tracking reveal temporal properties of attention. *Vision Research*, 40(26):3651–3664, 2000.

[64] R. E. Wray, J. E. Laird, A. Nuxoll, D. Stokes, and A. Kerfoot. Synthetic adversaries for urban combat training. In *Proceedings of the Sixteenth Conference on Innovative Applications for Artificial Intelligence*, July 2004.

[65] Y. Wu and T. S. Huang. Vision-based gesture recognition: A review. *Gesture-Based Communication in Human-Computer Interaction*, 1739:103–115, 1999.

[66] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Computing Surveys*, 38(4):1–45, December 2006.

[67] D. Zhang and B. Nebel. Learning a table soccer robot a new action sequence by observing and imitating. In *Proceedings of the Third Conference on Artificial Intelligence and Interactive Digital Entertainment*, pages 61–66, 2007.

Appendix A

Performance Measure Calculations

There are three steps involved in the computation of both the Prediction Accuracy Error (PAE) and Human Similarity Error (HSE).

Step 1: Compute the *frame error*.

For a given game frame t, the frame error $F(X_t, Y_t)$ is calculated as the mean error per prediction between the prediction vector X_t and reference vector Y_t. Specifically, Equation A.1 is used to calculate $F(X_t, Y_t)$ where π is a permutation over elements in Y_t and n_t is the number of opponents alive.[1] When calculating the PAE for a predictor, Y_t are the true positions of opponents at frame t. In contrast, calculating the HSE utilizes the predictions of human experts for Y_t.

$$F(X_t, Y_t) = \min_{\pi} \frac{1}{n_t} \left(\sum_{i=1}^{n_t} D(x_t^i, y_t^{\pi(i)}) \right) \tag{A.1}$$

The distance function $D(x, y)$ in Equation 6.1 is an approximate shortest path between coordinates $x, y \in \mathbb{R}^2$ in the game environment. Because the process of determining the shortest path between every pair of 3D coordinates is computationally prohibitive, the environment is partitioned by the visibility matrix and visibility cubes are used as vertices along the path. Specifically, the path is constructed from $C_V(x)$ to $C_V(y)$ through cube centers that have line-of-sight and are adjacent to each other in V (that is, all cubes sharing an edge, face, or corner with another). Because all predicted positions do not record a z-cooridnate value, the "column" of cubes $U(i)$ that exist above and below $C_V(i)$ is determined and the shortest path between any element of $U(x)$ and any element of $U(y)$ is used. Note that if coordinate i is placed in an invalid area of the environment (e.g., within a wall), $C_V(i)$ evaluates to the valid cube whose coordinate center is closest to i. In this way, it is impossible for a prediction to be placed "out of bounds".

Step 2: Compute the *game error*.

[1] The calculation in Equation A.1 is effectively a greedy minimal matching process. While more efficient algorithms exist for finding minimum weighted matchings (e.g., the Hungarian method [25]), matching 5 pairs of points requires only $5! = 120$ comparisons.

The game error G is calculated as the mean error per prediction over all frame errors for a game.[2] Equation A.2 shows how G is calculated where t represents the total number of frames in the game. Note that each frame error is weighted by the number of opponents alive during that frame. This prevents frames at the end of the game (which typically have few players left alive) from skewing the overall average.

Due to the short time span between game frames (0.45 seconds), the error calculated for a given frame can be influenced by the error for the previous frame. For instance, if a predictor made very poor predictions for frame t and no opponents were sighted by friendly players, it is quite likely that the performance for the next frame $t + 1$ will also be poor. Conversely, the frequent sightings resulting from large firefights would certainly help a predictor's accuracy over multiple frames. Despite this issue, the game error is still a desirable measure because situations such as these do exist in real-world scenarios and must be addressed by any competent prediction system.

$$G = \frac{\sum_t n_t \, F(X_t, Y_t)}{\sum_t n_t} \tag{A.2}$$

Step 3: Compute the *predictor error*.

Finally, the predictor error is computed as the mean error per prediction over all game errors from the set of testing logs.

[2] Only the error of every fifth frame is used when computing the HSE of a predictor to remain consistent with the frequency at which the humans predicted during the user study.

Appendix B

Detailed Experiment Results

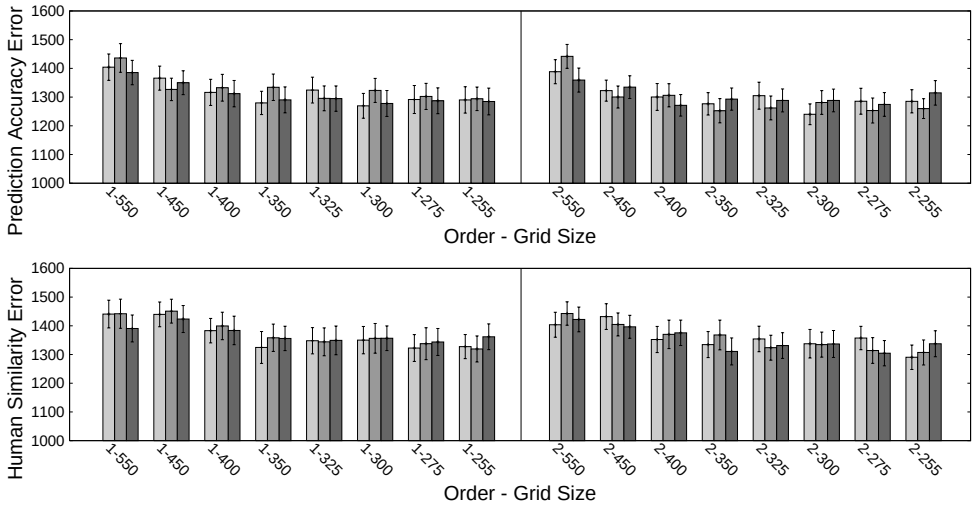


Figure B.1: Shared Training Data of PF(500) Terrorist Predictors. From left to right (i.e., light to dark), the bars represent the performance of predictors sharing "None", "Half", and "All" of the training data among the motion models.

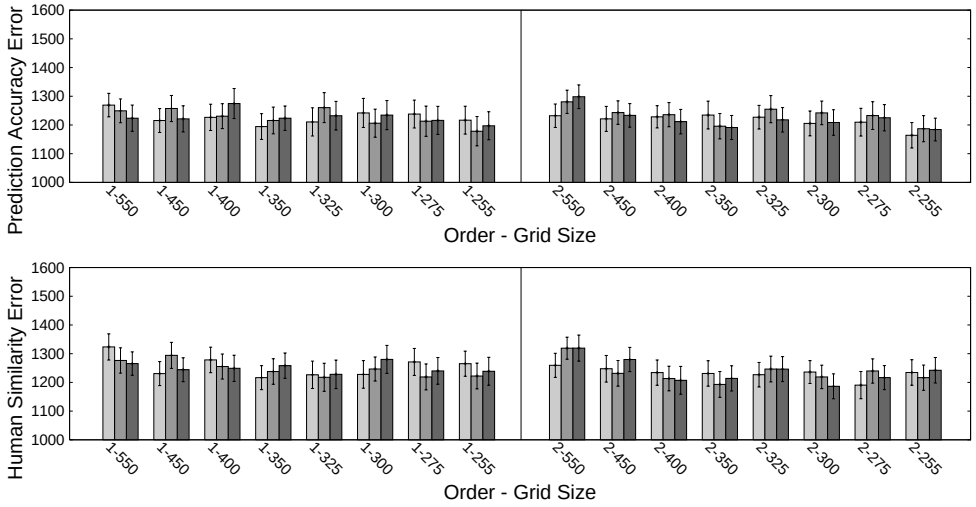


Figure B.2: Shared Training Data of PF(500) Counter-Terrorist Predictors. From left to right (i.e., light to dark), the bars represent the performance of predictors sharing "None", "Half", and "All" of the training data among the motion models.

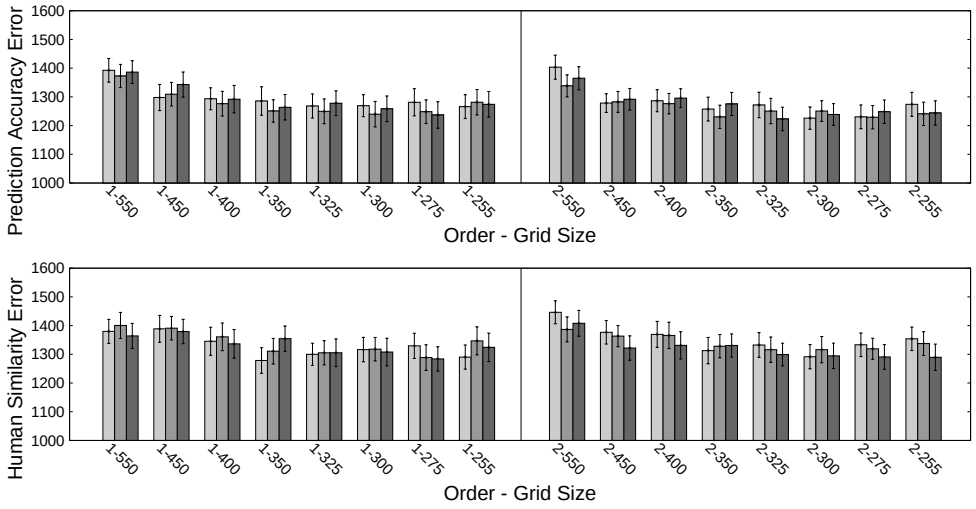


Figure B.3: Shared Training Data of PF(1000) Terrorist Predictors. From left to right (i.e., light to dark), the bars represent the performance of predictors sharing "None", "Half", and "All" of the training data among the motion models.

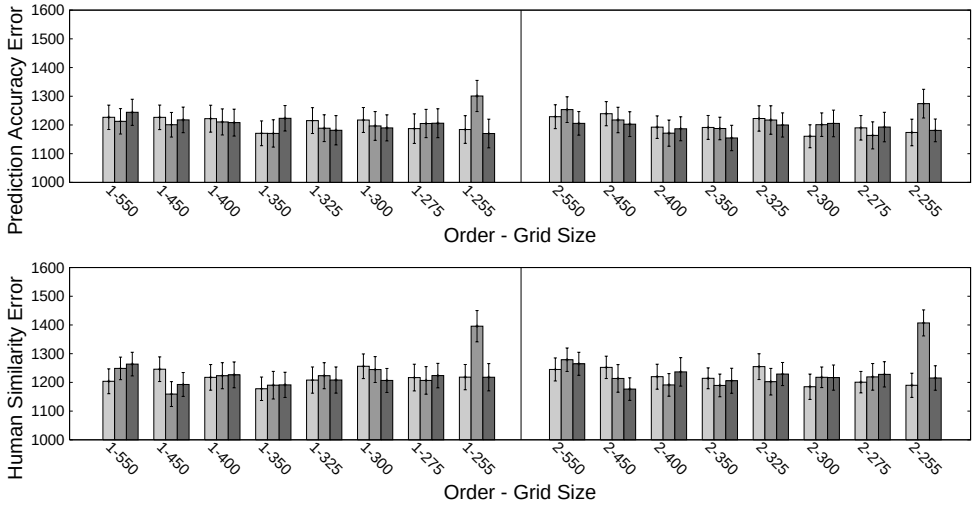


Figure B.4: Shared Training Data of PF(1000) Counter-Terrorist Predictors. From left to right (i.e., light to dark), the bars represent the performance of predictors sharing "None", "Half", and "All" of the training data among the motion models.

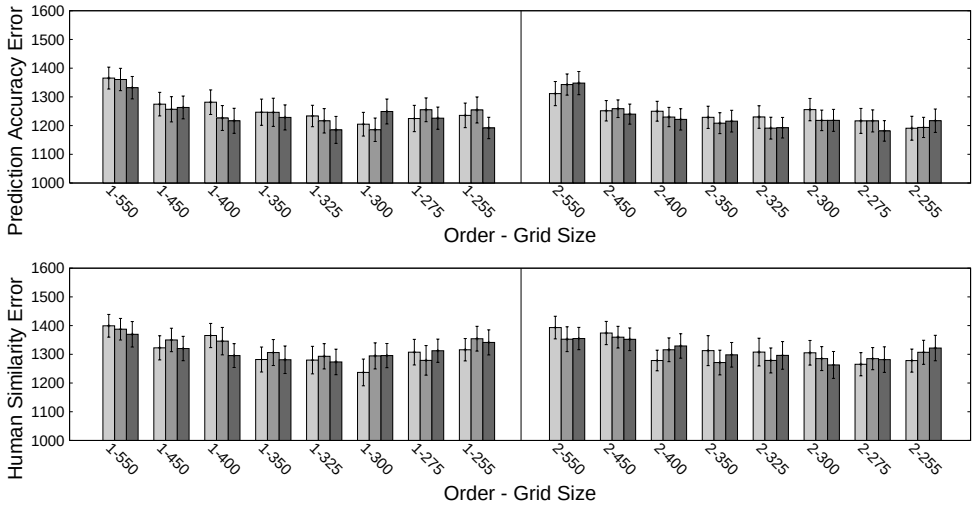


Figure B.5: Shared Training Data of PF(2000) Terrorist Predictors. From left to right (i.e., light to dark), the bars represent the performance of predictors sharing "None", "Half", and "All" of the training data among the motion models.

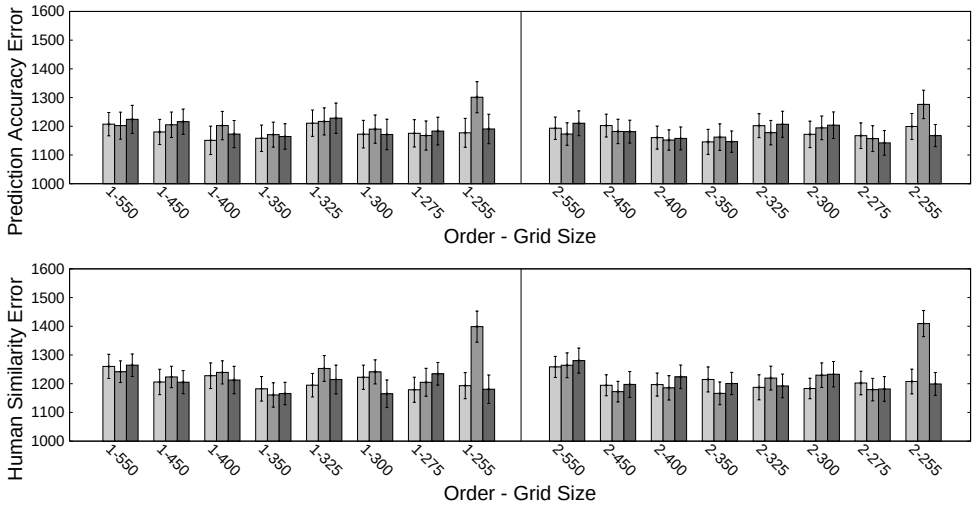


Figure B.6: Shared Training Data of PF(2000) Counter-Terrorist Predictors. From left to right (i.e., light to dark), the bars represent the performance of predictors sharing "None", "Half", and "All" of the training data among the motion models.

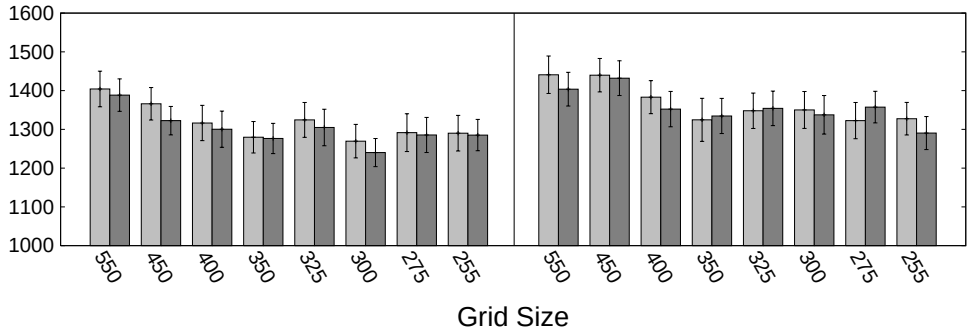


Figure B.7: First and Second-Order PF(500) Terrorist Predictors. The left image plots Prediction Accuracy Error along the y-axis while the right image plots Human Similarity Error. The light bars represent first-order predictors and the dark bars represent second-order.

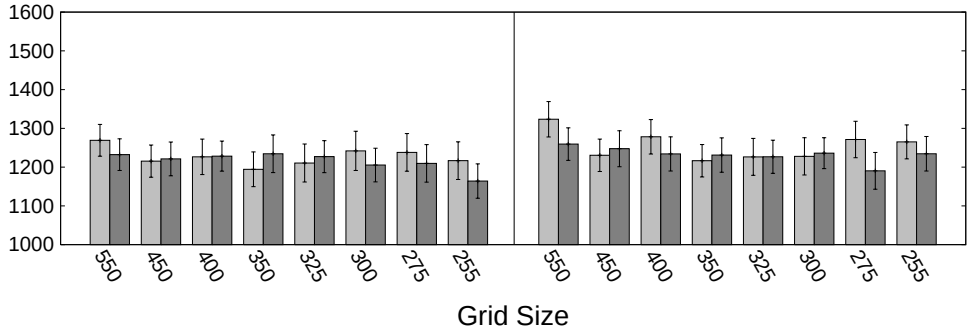


Figure B.8: First and Second-Order PF(500) Counter-Terrorist Predictors. The left image plots Prediction Accuracy Error along the y-axis while the right image plots Human Similarity Error. The light bars represent first-order predictors and the dark bars represent second-order.

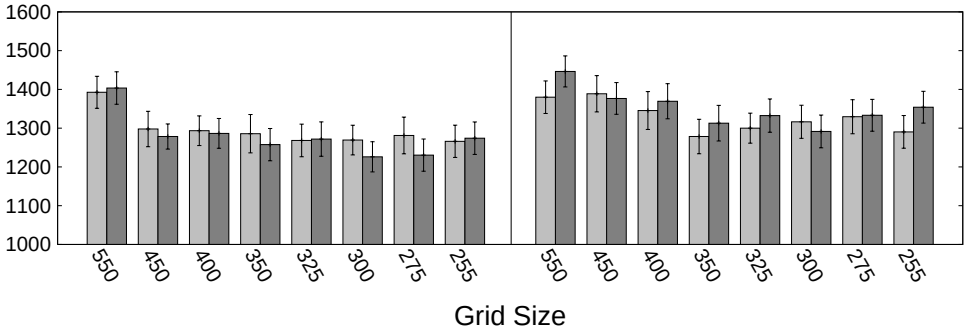


Figure B.9: First and Second-Order PF(1000) Terrorist Predictors. The left image plots Prediction Accuracy Error along the y-axis while the right image plots Human Similarity Error. The light bars represent first-order predictors and the dark bars represent second-order.

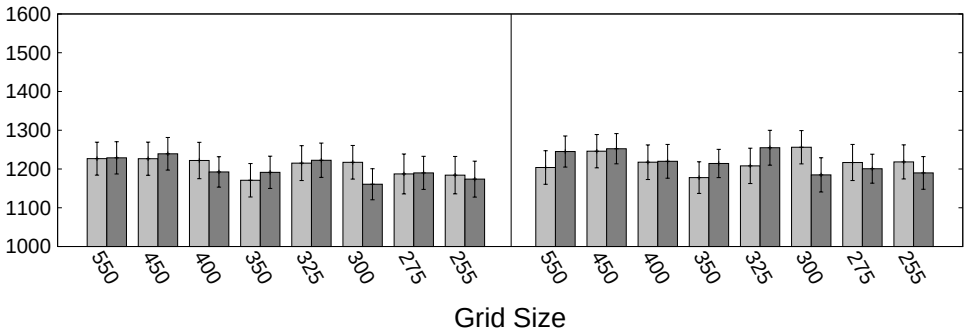


Figure B.10: First and Second-Order PF(1000) Counter-Terrorist Predictors. The left image plots Prediction Accuracy Error along the y-axis while the right image plots Human Similarity Error. The light bars represent first-order predictors and the dark bars represent second-order.

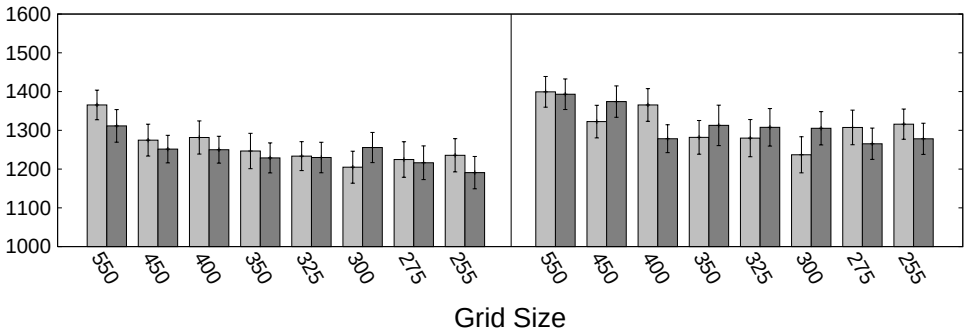


Figure B.11: First and Second-Order PF(2000) Terrorist Predictors. The left image plots Prediction Accuracy Error along the y-axis while the right image plots Human Similarity Error. The light bars represent first-order predictors and the dark bars represent second-order.

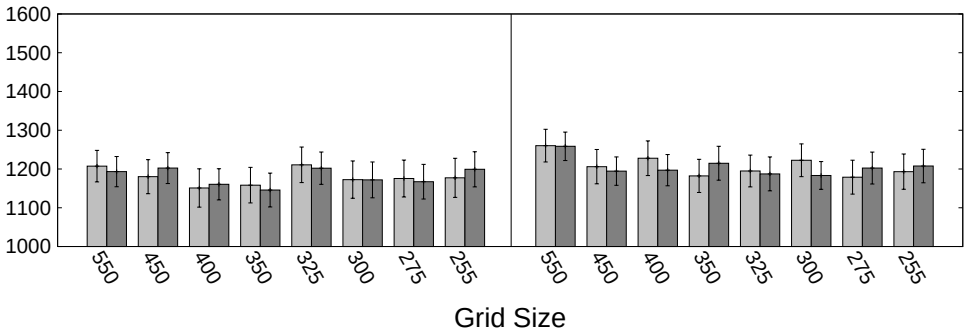


Figure B.12: First and Second-Order PF(2000) Counter-Terrorist Predictors. The left image plots Prediction Accuracy Error along the y-axis while the right image plots Human Similarity Error. The light bars represent first-order predictors and the dark bars represent second-order.

Appendix C

User Study Materials

UNIVERSITY OF ALBERTA
FACULTIES OF ARTS, SCIENCE & LAW
RESEARCH ETHICS BOARD

Application for Ethical Review of Human Research in the Faculty of Science

Instructions:

1. Use this form to request ethics review for research involving human subjects that does not require the use of identifiable health information. Human research that does involve identifiable health information should be submitted directly to the Health Research Ethics Board, http://www.hreb.ualberta.ca . Once the HREB review is completed, two (2) copies of the application together with HREB approval letter should be forwarded to the ASLREB Science member indicated below in (2).
2. Submit two (2) copies of this application form together with supporting materials (questionnaire instruments, interview questions, consent forms, recruitment materials, debriefing forms, safety approvals, etc.) to the ASLREB Science member, Dr. Tom Johnson, Department of Psychology, P-217 Biological Sciences Building.

`

A. Project Title: Evaluating Spatial Predictions in a Video Game Environment

B. Applicant Information

Name:	**Stephen Hladky**	E-Mail:	**hladky@cs.ualberta.ca**
Department:	**Computing Science**	Phone:	**780 492 2821**
Mailing Address:	**Department of Computing Science, ATH-221, University of Alberta**		

Are you: ☐ Faculty ☐ Staff ☒ Graduate Student ☐ Undergraduate Student

If you are a student or student intern:

Academic Supervisor:	**Vadim Bulitko**	E-Mail:	**bulitko@cs.ualberta.ca**
Department:	**Computing Science**	Phone:	**780 492 3854**

If you are a student intern:

Employment Supervisor:		E-Mail:	
Company/Institution:		Phone:	
Mailing Address:			

Other Investigators on this project

	Name	Institutional Affiliation /Department	E-mail address
1.	**Vadim Bulitko**	**Department of Computing Science**	**bulitko@cs.ualberta.ca**
2.			
3.			

C. Project Information

For the items below, please check all that apply:

Project Type: ☐ Staff ☐ Student ☐ Class Project ☐ Grant Proposal ☒ Thesis ☐ In Class Research ☐ Quality Assurance ☐ Secondary Analysis of Data ☒ Mass Testing ☐ Subject Pool

Funding: ☐ AHFMR ☐ CIHR ☐ NSERC ☐ SSHRC ☐ UofA Internal ☐ Other (specify):

D. Signatures

Your signature indicates that you agree to abide by all policies, procedures, regulations and laws governing the ethical conduct of research involving humans as described in GFC 66, **http://www.ualberta.ca/~unisecr/policy/sec66.html**

Applicant: _____ Date: _____

The signature of the supervisor(s) below indicates that the supervisor has reviewed and approved the student's proposal.

Academic Supervisor: _____ Date: _____

Employment Supervisor: _____ Date: _____

E. Project Details

1. Please provide a short summary of the project that describes the research objectives, principal methods employed, research participants, and hypotheses.

Providing real-life training in critical scenarios for humans is difficult because of expensive costs (e.g., piloting airplanes), potentially lethal dangers (e.g., military drills), and permanent consequences (e.g., diplomatic relations). Virtual Reality (VR) simulators are a viable alternative to real-life training because they present environments with low associated risks and costs while offering an opportunity for trainees to acquire essential skills. However, in order for VR simulators to be effective as a training tool, they must capture essential aspects of the training domain with high fidelity. In scenarios that require trainees to interact with other humans, such humans have to be controlled by intelligent computer programs with convincing realism. Traditional techniques for modelling complex human behaviour (e.g., if-then rules, finite state machines) have been met with little success. Our goal is to improve in this area by "machine learning" such behaviours from recordings of actual human behaviour. In particular, we are focusing on the ability of humans to predict the spatial locations of other agents within an enclosed environment. We have trained a computer program (the "predictor") to simulate this ability and we wish to compare its performance to that of human subjects.

The popular video game Counter-Strike: Source was chosen as a testbed because we have compiled a database of several hundred expert-level human gameplay logs, a crucial component for training the predictor. The game pits two opposing teams against one another, each with their own specific task to complete. The goal of the game is to prevent the opposing team from accomplishing its task, either by denying opponents opportunities to complete the task within a certain time limit, or by dispatching all opponents from the game. The predictor is trained to estimate the locations of players on a particular team. This user study will require participants to perform the same task, given the same information available to the predictor, such as the current time of the prediction and gameplay history.

The objective of this study is to evaluate the quality of the predictor by testing its predictions against human predictions. Our hypothesis is that, for a given gameplay log, our computer program will produce predictions for opponent locations that are as accurate or better than human predictions. Additionally, we wish to use the prediction data collected from the participants to train future prediction programs to perform more similarly to humans.

2. Describe the source of research participants. Indicate the manner in which participation will be solicited and the nature of any inducements or promises offered for participation. For secondary analysis of data, please describe the source and characteristics of the dataset.

Anyone of age 18 or older is able to volunteer his or her time to participate in the user study. A personal computer running the Windows operating system, an internet connection and an account on the user study website are required in order to participate. Advertisement mediums will consist of emails and posters within the university as well as posts on public websites and online forums. The user study will initially be restricted to at most 100 participants. However, if the research team deems it appropriate, this limit may be increased or dispensed entirely to accommodate more participants. No monetary compensation will be provided.

3. Describe the procedures to be used including the tasks and procedures involved in participating.

In order to register for the study, participants will be required to create an account on the user study website. To successfully create an account, the participant must first acknowledge and agree with the terms and conditions in the user study consent form. When an account has been created, the participant will be requested to complete an online questionnaire identifying their age, gender, experience with Counter-Strike: Source, etc. Also, participants will be asked to download a prediction-recording computer program. Instructions for use of the program will be provided through videos available on the user study website. Once the participant has downloaded the program to their computer, they will be able to run it by double-clicking on its icon. Upon doing so, the participant will be required to enter the username and password of their user study website account (Figure 1).

Next, a window will appear presenting a top-down view of a Counter-Strike: Source map (Figure 2) with a grid overlay. Arrows on the map indicate the orientations of visible players. The participant will be assigned to a particular team by the program. All arrows of players on the same team as the participant will be shown (in green) while opposing team arrows will remain hidden. The participant will be able to annotate the log by clicking and dragging their cursor over areas on the map where they predict players on the opposing team to be located (Figure 3); these predicted areas will be highlighted in blue.

The participant will be able to fast-forward and rewind through the gameplay. Additional features provided to the participant will include the "observations" of teammates. These areas, shown in red, denote the areas of the map that teammates are able to see (Figures 4 and 5).

Once a log has been fully annotated, the participant's predictions (i.e., the blue overlays) will be automatically uploaded to the user study website, and a new game log to annotate will be displayed. While there are no time limits imposed on the participant to complete an annotation, the average log takes approximately 5 to 10 minutes for an inexperienced person to annotate. The participant will be allowed to annotate as many logs as they choose to in a given session, and they may quit the program at any time. If the participant wishes to annotate more games, they may run the program again at their leisure (but will be required to enter in their username and password each time).

Participants will also be provided with standard website account services. These services include the ability to change one's password, delete one's account, and to specify which maps participants prefer to annotate.

4. Describe how you will deal with the issues of informed consent and continuing voluntariness of participation in the research. For minors, describe how you will obtain consent of guardians.

In order to create an account on the user study website, the participants will be required to read a consent form informing them of the nature and purpose of the user study as well as their rights as participants. Only those participants who fill out the form with valid information and select the check box will be registered in the user study. All participants will be informed that they may discontinue their participation at any time during the study. If a participant chooses to discontinue their participation, they will be able to delete their account through the user study website at which time all data collected from them will be deleted. Minors will not be allowed to participate in the user study.

5. Describe how you will grant anonymity to participants and how responses will be kept confidential. If names or other identifying information are coded with data, describe how access to data is limited and safeguarded. Indicate who will have access. If appropriate, describe how consent is obtained from participants for exceptions to anonymity/confidentiality (e.g., focus groups). If data are to be taken from existing sources, discuss the implications of pre-existing (implicit or explicit) guarantees of confidentiality/anonymity.

No names or other identifying information will be collected during the study. Each participant will be identified only by their user ID number which will be generated automatically by the user study website. While a username and password is required by each participant to log into the website and prediction-recording program, this information is stored in an encrypted format in a user study database, thus maintaining strict participant anonymity. Only members of the research team will have access to the data.

6. Describe your plans for the retention and disposal of data.

User study website access and all communications to and from the prediction-recording program will be through secure internet connections. All data will be stored in a secure database on a Computing Science department server for a minimum of five years. Access to the database will be allowed only to members of the research team.

7. If concealment and/or deception is to be employed, provide explicit justification. Indicate how and when participants will be informed of the concealment and/or deception.

No concealment or deception will be employed.

8. Describe the nature of any risks to the physical or psychological well-being or integrity of participants that might arise from your procedures, and discuss your justifications, safeguards, and resolutions for these risks where appropriate.

There will be no dangerous substances or risks of physical harm beyond those associated with a regular computer use. Users will be required to call upon prior experiences playing the game Counter-Strike: Source. Although Counter-Strike: Source has been rated Mature by the Entertainment Software Rating Board (see http://www.esrb.org/ratings/ratings_guide.jsp for details), the only original game assets the prediction-recording program uses is the top-down view of game maps (Figure 2). There are no visible or audible representations of violent content.

9. Indicate when participants will be debriefed, and describe the nature and extent of debriefing. Indicate how participants may follow-up with researchers to ask questions or obtain information about the study.

There will be no formal debriefing to participants because the nature and details of the study as well as the pretences under which it is being conducted will be made known prior to their participation. However, participants may access their user study website accounts during and after the conclusion of the study to reference an analysis of their prediction performance. This analysis will be similar to the example in Figure 6, a graph depicting three performance curves:

1. The performance of the participant on logs they have annotated.
2. The aggregate performance of all participants for the same set of logs.
3. The performance of the predictor or the same set of logs.

At any time participants can send an email to any member of the research team to ask any questions or raise any issues they might have.

10. Describe any apparatus, element of the physical environment, substance or other materials that could cause harm to a participant if a malfunction, misuse, accident, allergic reaction, or side-effect were to occur. If the participant comes into contact with a potentially hazardous apparatus or material, who will be responsible for checking for defects/malfunctions, and on what schedule will inspections be made? If participants come into contact with some substance that could cause harm, please document your safeguards. Describe safety approvals that you have obtained or applied for (e.g., biohazards, electromechanical, radiation, etc.)

Participants are required to use a computer with an internet connection to access the user study software and website. Effects from participating in the user study will be similar to that of home computer usage.

11. Describe qualifications of research personnel if special conditions exist within the research that could cause physical or psychological harm or if participants require special attention because of physical or psychological characteristics, or if made advisable by other exigencies.

No additional research personnel are required.

12. Describe any potentially hazardous duties that will be required of research personnel, including physical, mental, or legal risks. Describe the safeguards you have implemented for your personnel.

Not applicable.

Please submit twp (2) copies of your application together with supporting materials to Dr. Tom Johnson, Department of Psychology, P-217 Biological Sciences Building.

Figures Reference

FIGURE 1: Username and password entry

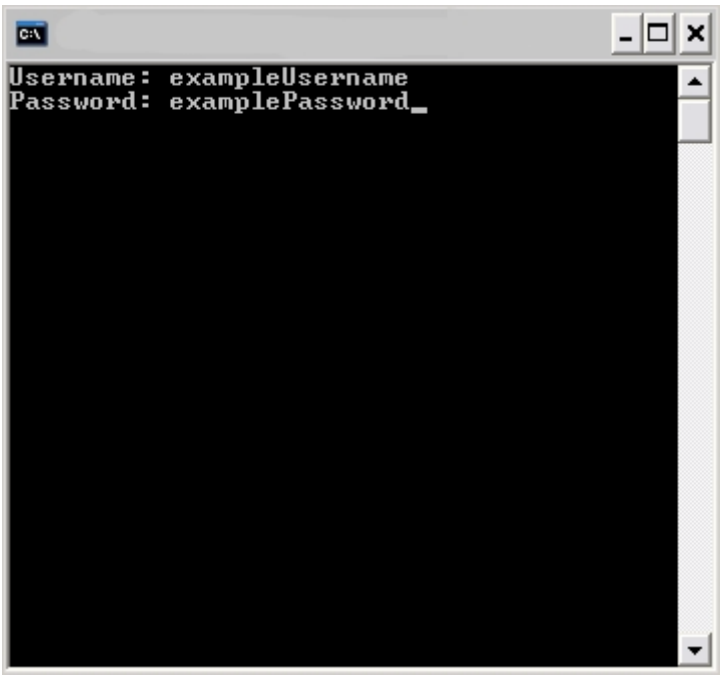


FIGURE 2: Main screen with grid overlay

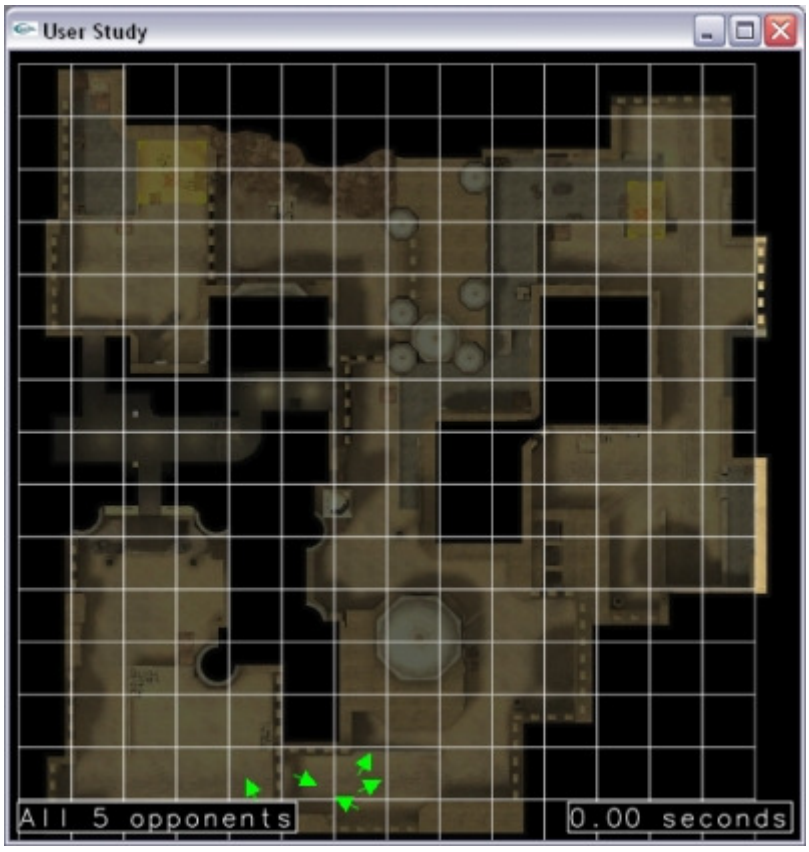


FIGURE 3: Main screen with a participant's prediction

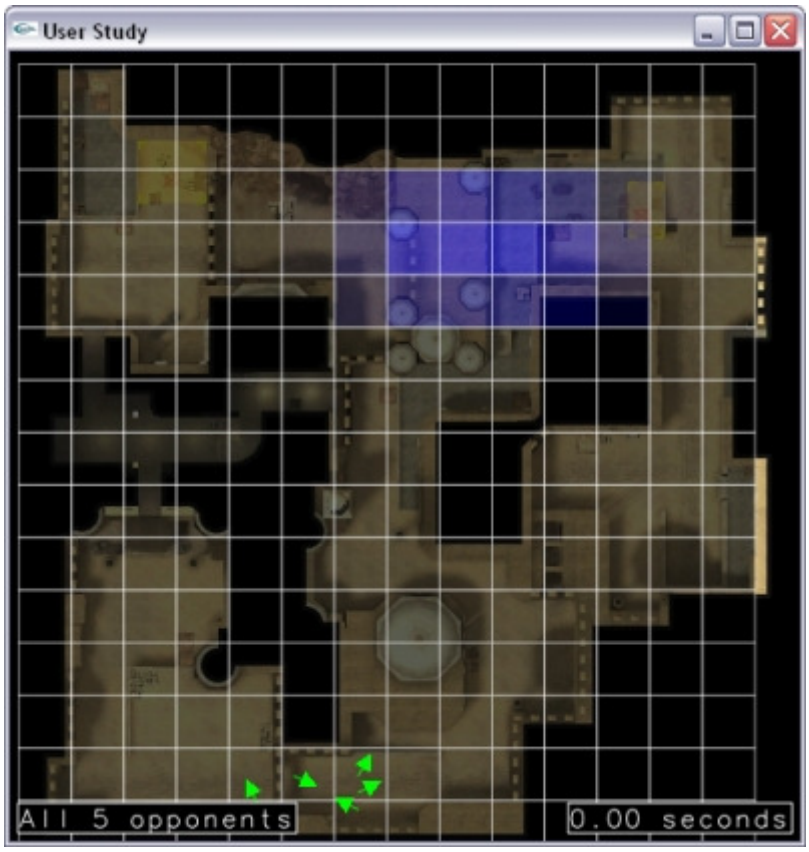


FIGURE 4: Observation overlay

FIGURE 5: Participant Predictions with Observation Overlay

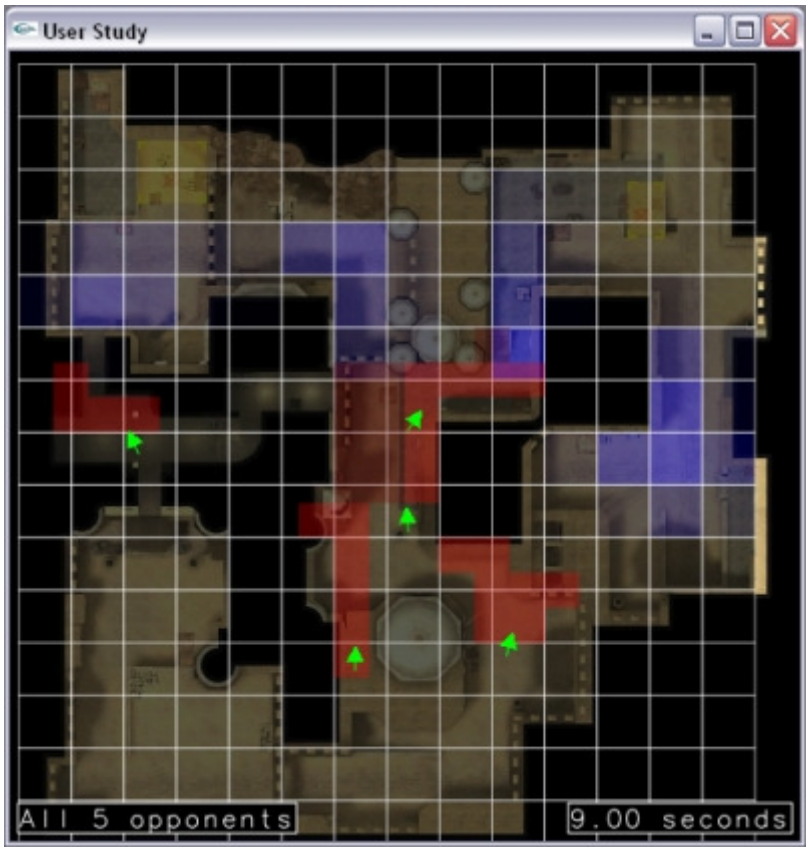
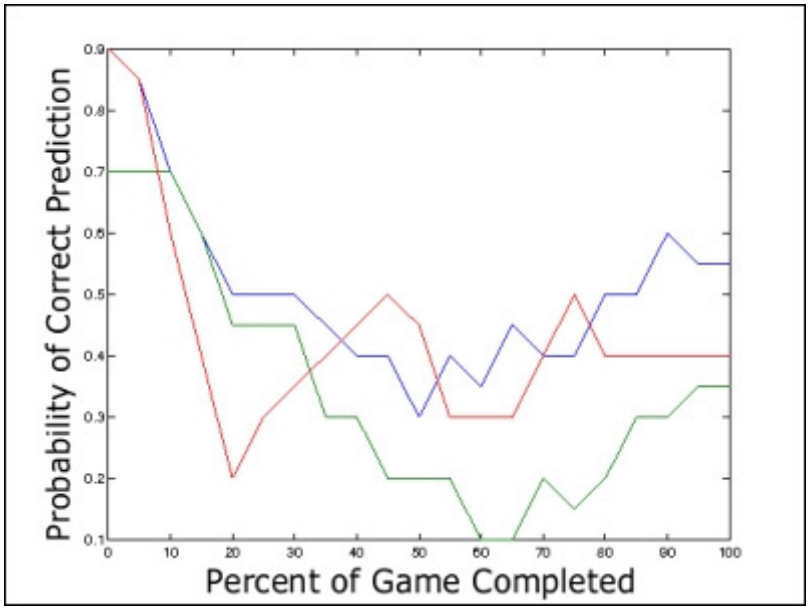


FIGURE 6: Example of a Participant's Performance



Research Information and Participant's Consent Form

You must be 18 years of age or older to participate in this study.

Introduction

Welcome! You are invited to participate in a research study being conducted by Stephen Hladky and Dr. Vadim Bulitko of the Department of Computing Science at the University of Alberta. The purpose of this study is to evaluate the ability of humans to predict the locations of opponents in the video game Counter-Strike: Source. There are two objectives to this study. First, we are interested in comparing human predictions with those made by computer programs to determine which group predicts with an overall higher accuracy. Secondly, we wish to use the prediction data from the participants to train prediction programs to perform more similarly to humans.

Your participation

In order to participate in the user study you are required to create an account on the user study website (http://ircl.cs.ualberta.ca/games/cs/userstudy). Through the account you must fill out an online questionnaire, which should take about five minutes. You may skip any item in the questionnaire by leaving them blank. You will also receive a link to a prediction-recording program, which you should download to your computer. Information on how to use the program can be found through instructional videos on the user study website.

When running the prediction-recording program, you will be presented with a replay of a Counter-Strike: Source match from a top-down view. Players on the map are represented by arrows indicating the directions that they are facing. The team composed of green arrows are friendly players; you will be able to see these arrows at all times. In contrast, the opposing team is composed of red arrows and remains hidden from view. You will be able to play the match using VCR-style forward and rewind controls, watching the arrows move throughout the map. At certain points in the replay, you will be required to annotate the map by providing a prediction of the locations of opponents. This is done by clicking your mouse on a grid to highlight grid cells that you believe to contain opponents.

To aid you in your predictions, you will be able to toggle an "observation overlay", a red-hue colour-map that indicates the locations on the map that friendly players are able to see. If an opponent is located at a position that your team can see, that opponent's arrow will become visible for that moment.

Once you finish annotating a game (which may take anywhere between five and ten minutes), your predictions will be uploaded to the user study web server and a new replay will be downloaded for you to annotate. Incomplete annotations will neither be saved nor sent to the user study server. You are free to skip any replay presented to you for any reason (e.g., you are unfamiliar with the current map, or feel you would not be able to

provide an accurate prediction for the current replay). You may annotate as many games as you wish; when complete, simply close the prediction-recording program.

You may annotate more games at later date by running the prediction-recording program at your leisure. Additionally, you will be able to view an analysis of your prediction performance via your user study account.

Your rights

Your decision to participate in this study is entirely voluntary and you may decide at any time to withdraw from the study. If you withdraw, all data collected from you during the study will be destroyed. All data collected will remain confidential and only researchers associated with the project will have access to the data. The data will be securely stored by Stephen Hladky for a minimum of five (5) years. The results of this study may be presented at scholarly conferences, published in professional journals, presented in class lectures, and used as part of or to inform future research projects. No monetary compensation will be provided to participants of the user study.

Benefits and risks

All Software (download-able programs and updates) and Services (facilities made available to participants through the user study, such as the user study website) are provided "as-is". The members of the user study research team, the Human Research Ethics Committee, the Department of Computing Science, and the University of Alberta shall not be responsible or liable, directly or indirectly, in any way for any loss or damage of any kind incurred as a result of, or in connection with your use of the Software and/or Services. Moreover, the user study research team reserves the right to at any time modify or discontinue, temporarily or permanently, all or part of the Software and/or Services as they deem appropriate. Such modifications or discontinuations will be preceded by a notice on the user study website. The Software and Services carry no warranties explicit or implied of any kind.

There are no foreseeable physical or mental risks to this study, but if any risks should arise, the researchers will inform the participants immediately. If you should experience any adverse effects, please contact Stephen Hladky immediately.

Contact information

If you have any questions of concerns on the study, or if you wish a clarification of your rights as a research participant, you can contact Stephen Hladky, Dr. Vadim Bulitko, or the Human Research Ethics Committee at the phone numbers or addresses below.

Stephen Hladky	**Vadim Bulitko, Ph.D.**	**Tom Johnson, Ph.D.**
M.Sc. Candidate	Assistant Professor	Chair, Human Research
Department of Computing	Department of Computing	Ethics Committee
Science	Science	Department of Psychology

University of Alberta
Edmonton, AB T6G 2E8
(780) 492-2821
hladky@cs.ualberta.ca

University of Alberta
Edmonton, AB T6G 2E8
(780) 492-3854

University of Alberta
Edmonton, AB T6G 2E9
(780) 492-2834

Signatures

By clicking the "Create Account" button below, you confirm that you have read and understood the purpose and details of the study and indicate your willingness to participate.

You must be 18 years of age or older to participate in this study.

Participant Questionnaire

PART I

(To be completed by the participant at the beginning of the user study (i.e. when a user account is created). Participants may change their answers at any time before the conclusion of the user study. Any answer may be left blank by the participant.)

1. What is your gender?

Male
Female

2. Please select your age group.

18 to 21 years
22 to 25 years
26 to 29 years
30 to 33 years
34 to 37 years
38+ years

3. How frequently do you play first-person shooter video games?

Never
Less than once a month
At least once a month
At least once a week
Daily

4. How frequently do you play Counter-Strike and/or Counter-Strike:Source?

Never
Less than once a month
At least once a month
At least once a week
Daily

5. How long have you been playing first-person shooter video games?

Less than 6 months
6 months to 1 year
1 year to 2 years
2 years to 4 years
4 years to 7 years
More than 7 years

6. How long have you been playing Counter-Strike and/or Counter-Strike:Source?

Less than 6 months
6 months to 1 year
1 year to 2 years
2 years to 4 years
4 years to 7 years
More than 7 years

PART II

(To be completed by the participant after at least one game has been annotated. Participants may change their answers at any time before the conclusion of the user study. Any answer may be left blank by the participant.)

7. Please describe any techniques, procedures, reasoning, or thoughts that you felt helped you to predict opponent locations.

_____
_____
_____

8. Please record any comments you may have about the user study (e.g., how it was conducted, overall experience, etc.)

_____
_____
_____