ABSTRACT

Title of dissertation: MODEL-PREDICTIVE STRATEGY GENERATION FOR MULTI-AGENT PURSUIT-EVASION GAMES

Eric Raboin, Doctor of Philosophy, 2015

Dissertation directed by: Professor Dana Nau
Department of Computer Science

Multi-agent pursuit-evasion games can be used to model a variety of different real world problems including surveillance, search-and-rescue, and defense-related scenarios. However, many pursuit-evasion problems are computationally difficult, which can be problematic for domains with complex geometry or large numbers of agents. To compound matters further, practical applications often require planning methods to operate under high levels of uncertainty or meet strict running-time requirements. These challenges strongly suggest that heuristic methods are needed to address pursuit-evasion problems in the real world.

In this dissertation I present heuristic planning techniques for three related problem domains: visibility-based pursuit-evasion, target following with differential motion constraints, and distributed asset guarding with unmanned sea-surface vehicles. For these domains, I demonstrate that heuristic techniques based on problem relaxation and model-predictive simulation can be used to efficiently perform low-level control action selection, motion goal selection, and high-level task allocation.

In particular, I introduce a polynomial-time algorithm for control action selection in visibility-based pursuit-evasion games, where a team of pursuers must minimize uncertainty about the location of an evader. The algorithm uses problem relaxation to estimate future states of the game. I also show how to incorporate a probabilistic opponent model learned from interaction traces of prior games into the algorithm. I verify experimentally that by performing Monte Carlo sampling over the learned model to estimate the location of the evader, the algorithm performs better than existing planning approaches based on worst-case analysis.

Next, I introduce an algorithm for motion goal selection in pursuit-evasion scenarios with unmanned boats. I show how a probabilistic model accounting for differential motion constraints can be used to project the future positions of the target boat. Motion goals for the pursuer boat can then be selected based on those projections. I verify experimentally that motion goals selected with this technique are better optimized for travel time and proximity to the target boat when compared to motion goals selected based on the current position of the target boat.

Finally, I introduce a task-allocation technique for a team of unmanned sea-surface vehicles (USVs) responsible for guarding a high-valued asset. The team of USVs must intercept and block a set of hostile intruder boats before they reach the asset. The algorithm uses model-predictive simulation to estimate the value of high-level task assignments, which are then realized by a set of learned low-level behaviors. I show experimentally that using model-predictive simulations based on Monte-Carlo sampling is more effective than hand-coded evaluation heuristics.

MODEL-PREDICTIVE STRATEGY GENERATION FOR MULTI-AGENT PURSUIT-EVASION GAMES

by

Eric Raboin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2015

Acknowledgments

This dissertation would not have been possible without the generous support and mentorship of my advisor Dana Nau, whose advice and patience I am supremely grateful for. I am also grateful for the support provided by Satyandra K. Gupta and Petr Švec, whose guidance helped shape many of the contributions in this work. Finally, I am eternally grateful for my family and friends, who have been an unwavering source of confidence and encouragement throughout many years of study.

This work was also supported in part by ONR grants N000141310597 and N000141210430. The information in this paper does not necessarily reflect the position or policy of the funders, and no official endorsement should be inferred.

Table of Contents

Chapter 1: Introduction

A wide variety of practical and theoretical problems in robot motion planning can be thought of as pursuit-evasion problems. Although originally formulated to model "predator" and "prey" type scenarios, pursuit-evasion problems can be extended to cooperative situations, or situations where the objective is to observe or block an evader. Many potential applications exist: robotic security guards patrolling buildings to detect intruders, teams of robots conducting remote search-and-rescue missions, assembly robots tracking human workers on the factory floor, or unmanned boats defending shipping vessels from hostile incursion.

The attraction of pursuit-evasion games in literature has yielded a variety of novel algorithms and theoretical results [1–9]. However, fundamental challenges still exist for solving pursuit-evasion problems efficiently. $\mathcal{NP}$-hardness results have been established for pursuit-evasion problems in both discrete [1] and continuous domains [2,3]. As a result, prior research has traditionally focused on solving relaxed versions of the problem [4–6], or precomputing solutions to the problem offline [7–9]. However, many of these algorithms have running times that grow exponentially as the size of the domain or number of agents increase, rendering them unsuitable for situations where planning must be done quickly.

In this dissertation I explore how problem relaxation and model-predictive simulation can be used to develop computationally efficient planning heuristics for pursuit-evasion problems. In particular, this work introduces novel methods that incorporate Monte-Carlo sampling and learned probabilistic opponent models to efficiently evaluate candidate action plans in several problem domains.

This work provides algorithmic contributions in the following three areas:

- A control action selection algorithm for visibility-based pursuit-evasion games where a team of pursuer agents must track an evader through two-dimensional environments with partial visibility
- A motion goal selection algorithm for unmanned boats subject to differential motion constraints where a pursuer boat must follow an evader boat through an environment with obstacles
- A contract-based task allocation algorithm for a team of unmanned sea-surface vehicles (USVs) which must intercept and block hostile intruder boats before they reach a high valued asset

This work is organized into four chapters, described below.

Chapter 2 introduces an efficient planning heuristic for visibility-based pursuit-evasion games. In these games, the exact location of the evader is not known unless it is within the limited sensor range of a pursuer, allowing the evader to move behind obstacles or outside of sensor range to evade detection. The objective of the pursuer team is to minimize uncertainty about the evader's location. The planning heuristic evaluates control actions for the pursuer team by performing lookahead on a relaxed version of the problem and estimating the potential for future visibility loss. This approach scales linearly as the size of the domain or number of agents increases. Experimental results show that the algorithm performs better than naive approaches which follow the shortest path to the evader. The chapter also discusses how this approach accounts for interrupted communication between agents.

Chapter 3 extends the planning heuristic from the previous chapter to support probabilistic opponent models. Particle filtering is used to estimate the posterior distribution of evader locations given the observation history of the pursuer team. Particle movements are simulated using Monte-Carlo sampling over a weighted potential field, which may be learned from interaction traces of prior games. Results show that significant performance gains are seen over the non-probabilistic heuristic even if a naive, random opponent model is used. When the learned model is used, the heuristic obtains faster recovery times than competing algorithms based on worst-case analysis.

Chapter 4 introduces motion goal selection algorithm for pursuit-evasion scenarios involving two boats subject to differential motion constraints. The objective of the pursuer boat is to position itself behind the evader while safely avoiding obstacles. Monte-Carlo sampling is used to generate a posterior distribution over the future location and orientation of the evader boat. The resulting distribution is used to select candidate motion goals for the pursuer. Experimental results show that motion goals selected using this technique reduce the pursuer's travel time and increase the percentage of time the pursuer is within range of the evader when compared to techniques that use the current evader location as a motion goal.

Chapter 5 introduces a contract-based task allocation algorithm for a team of unmanned sea-surface vehicles (USVs) responsible for guarding a high-valued asset against incursion by hostile intruder boats. The team of USVs must position themselves around the asset which is located in high-traffic area frequented by passing civilian boats. When one or more intruder boats are identified, the USV team must assign responsibility to intercept and block the intruders, delaying their approach to the asset for as long as possible. The task allocation strategy uses Monte-Carlo sampling and model-predictive simulation to evaluate candidate task exchanges, accounting for uncertainty about the identity of the intruder boats and the opponent model used by the intruder. Experimental results show that model-predictive simulation is more effective at performing task-allocation than hand coded heuristic rules. The performance of the strategy can be improved by increasing the number of simulations performed and expanding the set of candidate task allocations evaluated. Once tasks are assigned, they are realized by low-level behaviors that are have been optimized using a genetic algorithm. Details about the low-level behaviors used by this algorithm are provided in Appendix A.

Chapter 2: Visibility-Based Pursuit-Evasion

This chapter introduces a heuristic planning technique for visibility-based pursuit-evasion games, where a team of pursuer agents must track an evasive target through an environment with obstacles that obstruct visibility. When the evader passes behind an obstacle, the pursuer team no longer knows its exact location. The objective of the pursuer team is to minimize uncertainty about the location of the evader by reducing the likelihood of visibility loss and recovering from it effectively when it occurs. Potential applications of this problem include surveillance, search and rescue, and environmental monitoring.

Almost all traditional instances of the visibility-based pursuit-evasion problem are computationally difficult. Identifying if a particular domain is solvable with a given number of pursuers is provably $\mathcal{NP}$-hard [2] and existing solution techniques have running-time complexities that are exponential in both the size of the domain and the number of agents [9, 10]. This holds true even when making simplifying assumptions, such as planning only for cases where the evader remains visible [11], or treating the evader as if it has unbounded speed [7].

To function effectively in the real-world, the pursuer team must react to changes quickly or risk permanently losing the evader. This means that control actions for the pursuer team must be generated efficiently, despite the computational challenges that are inherent to the problem. Additionally, any effective planning technique should ideally be robust against sensor and communication uncertainty and generalize to a wide range of environments.

This chapter introduces limited Euclidean-space lookahead (LEL), a control action selection heuristic for visibility-based pursuit evasion games satisfying the practical challenges described above. This algorithm generates sub-optimal but effective strategies in polynomial time by utilizing problem relaxation and generating simplified projections about each of the agents possible movements. The primary features of LEL are its low computational complexity and its applicability to scenarios where communication between agents may be interrupted. The algorithm is also able to generate control actions when the evader is either visible or hidden from view, a feature not supported by many previous approaches.

The following section provides a brief overview of prior research on the visibility-based pursuit-evasion problem, while Sec. 2.2 offers a formal definition of the problem approached in this chapter. The remainder of the chapter is spent describing the LEL heuristic and detailing experimental results that evaluate its performance against three different evader strategies.

2.1 Background

This section provides a review of existing literature on visibility-based pursuit-evasion games, as well as graph searching, a related pursuit-evasion formalism for discrete graphs.

2.1.1 Graph Searching

Graph-searching, a problem closely related to visibility-based pursuit-evasion, has long been a topic of interest in algorithms literature [12–26]. In graph-searching, a team of pursuers must navigate some graph G, where nodes represent locations and edges represent possible moves. The objective of the pursuers is to capture an evader whose movement is also restricted to the graph. Various rules have been defined for what determines a capture state, how agents are allowed to move, and what information pursuers have about the evader [12, 13].

In the original formulation by Parsons [14], no upper bound is placed on the speed of the evader. The problem is modeled after rescuers exploring a cave in search of a lost spelunker, where the objective of the pursuer team is to traverse the graph in such a way that all possible paths for the evader are guaranteed to be crossed. The *search number* of a graph is the minimum number of pursuers required to search the graph and successfully find the evader. Analysis by Megiddo et al. [15] showed that identifying the search number for Parsons' problem is $\mathcal{NP}$-complete for arbitrary graphs, but solvable in linear-time for trees.

Another version of the graph searching problem is where pursuers and evaders takes turns moving in alternating rounds. This formulation often goes by the name "cops and robbers", due to parallels with the real-world problem of tracking a fugitive [16,17]. The inclusion of alternating rounds places an upper bound on the speed of the evader, adding an extra level of complexity as pursuers are forced to reason about the evader's strategy. Identifying the search number of a graph in "cops and robbers" is known to be EXPTIME-complete [18] even if the location of the evader is known throughout the game. If the location of the evader is not known, or if agents are allowed to move simultaneously, then a non-deterministic strategy for the pursuer team is required [13, 19]. Research into the non-deterministic version of the problem has focused on calculating lower bounds for the number of pursuers and the number of steps required to capture the evader [19–21].

Numerous other versions of the graph searching problem exist, including games where agents traverse a directed graph [22, 23], where pursuers have a fixed capture radius [24], or where the pursuers can travel arbitrarily between any two nodes on the graph [25, 26]. In the latter problem, pursuers are said to travel by "helicopter," an extension of the cops and robbers analogy. A common characteristic across all graph searching problems is that motion is restricted to a graph, a constraint that is relaxed in visibility-based pursuit-evasion.

2.1.2 Visibility-Based Pursuit-Evasion

Visibility-based pursuit-evasion games are a special category of pursuit-evasion games where the pursuer team is defined to have limited sensor capabilities, meaning only portions of the environment can be observed at any given time. Unlike graph searching, the objective of the pursuer team is not to capture the evader, but to acquire or maintain visibility. The problem domain is also no longer restricted to discrete graphs, and is frequently represented as a two-dimensional plane or polygon.

The majority of the existing literature on visibility-based pursuit-evasion falls into one of two categories:

- Visibility-based pursuit-evasion games where the objective of the pursuers is to maintain visibility on the evader for as long as possible [27–30].
- Visibility-based pursuit-evasion games where the objective of the pursuers is to detect an evader that was previously hidden from view [2, 7, 10, 31], also called "hider-seeker" games.

$\mathcal{NP}$-hardness results have been obtained for both of the above versions of the visibility-based pursuit-evasion problem [2, 11]. The former focuses on *perfect-information* scenarios, where the state of the world is known exactly. The latter focuses on *imperfect-information* scenarios, where the exact location of the evader is not revealed until the end of the simulation. The work in this dissertation is more closely related to the second category, since the location of the evader is not always known. However, prior research does not typically generalize to scenarios where the evader passes both in and out visibility, something which is directly addressed by the work introduced in this chapter. Although this limits direct comparison with existing approaches, a brief summary of the related literature is provided below.

There are several existing approaches to the problem of maintaining visibility on the evader [27–30]. These include generating paths for the pursuer to maximize visibility on a evader whose trajectory is known in advance [27], tracking a evader through an unknown environment by modeling the evader's possible escape paths

[28], and maintaining visibility at a fixed distance [29,30]. Provably optimal methods have also been developed for discrete domains using backwards induction [32], and game theoretic analysis has proven the existence of Nash equilibria for continuous versions of the problem [33]. In general, these approaches do not address what happens after visibility on the evader has been lost, ignoring any preemptive actions that should be performed to increase the chance of recovery. The optimal solution methods also suffer from computational tractability issues, scaling exponentially as the size of the domain or number of agents increases [32, 33].

The work of LaValle et al. [2, 7] defines the most well-known approach to visibility-based pursuit-evasion problems where the evader is hidden from view. In this formulation, the pursuer and evader are restricted to a two-dimensional polygonal environment, and the objective of the pursuer team is to search the polygon in such a way that the evader is guaranteed to be detected. The algorithm performs an information-space search by discretizing the problem space along critical information boundaries, such as bitangent rays and inflection rays, where the states contained within each discretized region can be safely aggregated.

The algorithm introduced by LaValle et al. is guaranteed to find a solution path if one exists, and the approach has been extended to solve problems where the evader's speed is bounded [9] and where the pursuer has a limited field-of-view [10]. However, each of these extensions has significant penalties for the algorithm's performance, as the complexity of the algorithm grows exponentially as the density of critical information boundaries increases. Due to this issue, the approach has not yet been extended to three-dimensional domains [5], and heuristic techniques have been proposed for dealing with multiple pursuers, such as moving only a single pursuer at any given time [34]. Since the algorithm by LaValle et al. does not specify what to do in situations where the evader is visible, it is not directly comparable to the approach outlined in this chapter. However, we do provide an experimental comparison with the LaValle algorithm for recovery scenarios in Chapter 3.

The work by LaValle et al. additionally demonstrate that, for domains with certain topology, unless a minimum number of pursuers is provided, no sequence of actions can be performed that is guaranteed to detect the evader [7]. This is a consequence of the "worst-case" assumption made about the evader's movement, and applies equally to all algorithms which make the same assumption, including the heuristic evaluated in this chapter. This leads to scenarios where the evader is unrecoverable, which has consequence for the performance of the algorithm in this chapter. However, we show how this limitation can be relaxed in Chapter 3, by providing a probabilistic model of the evader.

The relaxed lookahead (RLA) heuristic, developed as part of the preliminary work for this dissertation, is capable of evaluating strategies for a simple gridworld version of the visibility-based pursuit-evasion problem [35]. The RLA heuristic works by projecting the future location of the evader and pursuers using a relaxed version of the problem. The *limited Euclidean-space lookahead* (LEL) heuristic introduced in this chapter is an extension of RLA to problems in continuous Euclidean space, which is robust against interruptions in communication between agents [36]. Details about RLA are provided in Sec. 2.3.

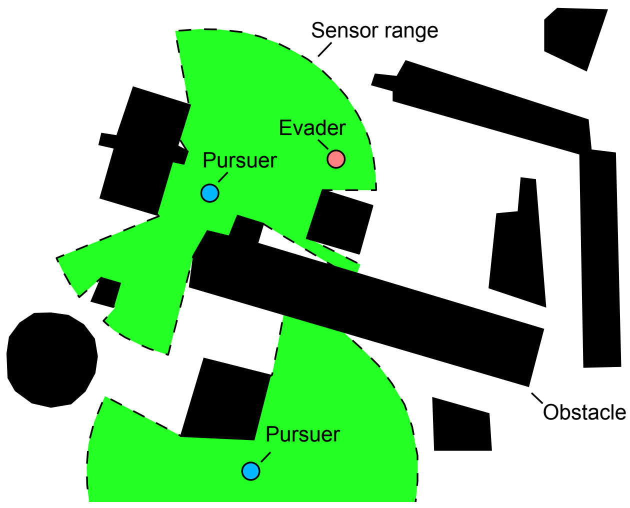


Figure 2.1: Example pursuit scenario with two pursuer agents and a single evader agent. Shaded areas represent the region that can be observed by the pursuer agents.

2.2 Problem Formulation

This chapter defines a multi-agent, imperfect-information game where a single evader agent a_0 is pursued by a team of n pursuer agents $\{a_1, a_2, \ldots a_n\}$. The goal of the pursuer team is to minimize its uncertainty about the evader's location by the end of the game. The degree of uncertainty is determined by the size of the *contamination region*, the minimum set of locations guaranteed to contain the evader based on the the pursuer team's observation history. Agents' movement and observation capabilities are formally defined below.

2.2.1 States and Histories

We assume that each agent a_i is a holonomic point robot with a fixed maximum speed v_i. Agents can be located anywhere in the region $L_{free} \subseteq \mathbb{R}^2$, defined as free space. The domain may have multiple obstacles, where each obstacle is a polygon in $\mathbb{R}^2$, and L_{free} is the set of locations not intersecting any obstacles.

The game's state $s \in S$ is a set of locations for each agent, $\{l_0, l_1, \ldots l_n\}$, and a time t_k. Each game has an initial state s_0, and a time t_{end} indicating when the game ends. A game's history $h \in H$ at time $t_k \leq t_{end}$ is the set of paths followed by each agent $\{f_0, f_1, \ldots f_n\}$ from time t_0 until t_k, where $f_i(t)$ denotes the location of agent a_i at time t. Since agents can move freely in two-dimensional Euclidean space, the set of all states S, and set of all game histories H, are both infinite.

2.2.2 Reachability and Visibility

Agent a_i can travel from location l_j to location l_k only if a path exists from l_j to l_k, and every location in that path is contained in L_{free}. Thus, agent a_i's *reachability* function is

$$\begin{aligned} R_i(l_j, t) = \{l_k : &\text{ locations } \langle l_j, l_k \rangle \text{ are connected in } L_{free} \\ &\text{by a path of length } d(l_j, l_k) \leq t v_i\} \end{aligned}$$

which is the set of locations agent a_i can reach in time t starting from location l_j. This can be generalized to $R_i(L, t) = \{l_k : l_j \in L \wedge l_k \in R_i(l_j, t)\}$ which is the set of locations agent a_i can reach in time t starting from anywhere in $L \subseteq \mathbb{R}^2$.

Agent a_i can observe location l_k from location l_j only if l_k is contained within the visible region $V_i(l_j)$. The visible region is defined as the set of locations within a_i's sensor range, r_i, where the line-of-sight is not obstructed by an obstacle. Thus, agent a_i's *visibility* function is

$$V_i(l_j) = \{l_k : \text{locations } \langle l_j, l_k \rangle \text{ are connected in } L_{free}$$

$$\text{by a straight line segment of length } d \leq r_i\}$$

which is the set of locations visible to agent a_i while located at l_j. Visibility can also be generalized as $V_i(L) = \{l_k : l_j \in L \wedge l_k \in V_i(l_j)\}$ which is the set of locations agent a_i can observe while located somewhere in $L \subseteq \mathbb{R}^2$. An example state of the game that illustrates pursuers' visibility is shown in Fig. 2.1.

Agent a_i may recall its past location $f_i(t)$ for any time $t \leq t_k$, but it does not know the path f_j followed by any other agent $a_{j \neq i}$. Agent a_i must infer the location of the other agents based on the initial state s_0 and its observation history.

2.2.3 Observation Histories

During the game, each agent a_i's *observation history* is a finite set of observations $O_i = \{o_0, o_1, \ldots o_k\}$ where each observation is a tuple $\langle a_j, L, t \rangle$, meaning $f_j(t) \in L$, or "agent a_j is located in region L at time t." If observation o appears in agent a_i's observation history at time t, then the information in o is available to agent a_i at any time $t' \geq t$.

Observations are made at discrete time intervals, such that the number of observations in a particular observation history remains finite. Since agents are free to move between observations, we define a set of rules for computing the possible paths followed by each agent that are consistent with prior observations.

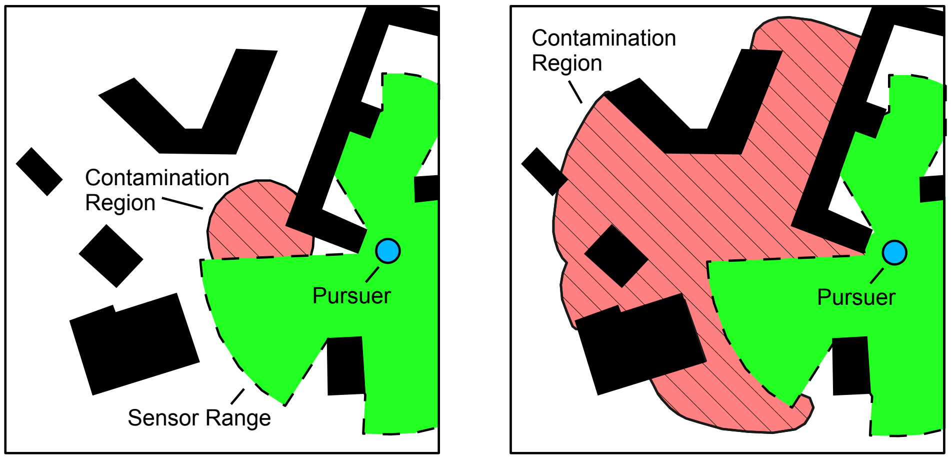


Figure 2.2: (left) Example scenario where a single pursuer has lost sight of an evader. The shaded area represents the contamination region, which is the set of locations that may contain the evader. (right) Larger area that results from expanding the boundary of the contamination region.

Given an observation history, an agent is able to calculate the region where another agent may be located, even if the actual location is not known. Given O_i, the set of locations guaranteed to contain agent a_j at time t is

$$R_j^+(O_i, t) = R_j(L, t - t') \tag{2.1}$$

where $\langle a_j, L, t' \rangle$ is the most recent observation in O_i describing agent a_j at some time $t' \leq t$. This expands the set of locations where a_j might be, also known as the contamination region, as illustrated in Fig. 2.2.

If agent a_i observes agent a_j at time t, meaning $f_j(t) \in V_i(f_i(t))$, then the tuple $\langle a_j, \{f_j(t)\}, t \rangle$ is added to agent a_i's observation history. However, if a_i does not directly observe a_j, the observation history is updated with a projected set of locations instead, defined as

$$\textit{projected}_j(O_i, f_i, t) = R_j^+(O_i, t) \setminus V_i(f_i(t)) \tag{2.2}$$

which is the set of locations that a_j can reach by time t, minus the locations observed by agent a_i. If a_i does not directly observe a_j at time t, then $\langle a_j, \textit{projected}_j(O_i, f_i, t), t\rangle$ is added to agent a_i's observation history.

In addition to its own observations, each pursuer receives periodic updates from the other agents on their team. An update from pursuer a_j informs the other pursuers of a_j's current location, as well as a_j's observation history O_j. This can be merged with a_i's latest observations by computing

$$\textit{merge}_k(O_i, O_j, t) = R_k^+(O_i, t) \cap R_k^+(O_j, t) \tag{2.3}$$

where $\langle a_0, \textit{merge}_0(O_i, O_j, t), t\rangle$ represents a_i and a_j's combined knowledge of the evader at time t. This observation, and $\langle a_j, \{l_j\}, t\rangle$ are both added to pursuer a_i's observation history as the result of the update.

Agent a_i's observation history O_i and path f_i map to an information set $I_i(t) \subseteq H$, which is the set of possible game histories given a_i's knowledge at time t. History h is in $I_i(t)$ if and only if $\langle f_i, O_i\rangle$ is consistent with h. Formally,

$$I_i(t) = \{h : (f_i \in h) \wedge \forall_{f_j \in h} Z(O_i, f_j)\}$$

where $Z(O_i, f_j) = \langle a_j, L, t\rangle \in O_i \to f_j(t) \in L$ implies that f_j is consistent with observation history O_i. As with states and histories, the set of all possible information sets at time $t > t_0$ is infinite.

In practice, we only require the most recent observation from each history. This is sufficient both to compute the LEL heuristic introduced in Sec. 2.3 and to maintain an accurate contamination region for the evader. Therefore, older observations may be safely discarded in the implementation.

2.2.4 Strategies

A *pure strategy* σ_i for agent a_i is a function mapping the agent's information set, $I_i(t)$ to the move it should perform at time t. Since changes to agent a_i's observation history occur only at regular time intervals, $\sigma_i(I_i(t))$ should specify a path f for agent a_i to follow from time t until the next update occurs to O_i. Path f is feasible for a_i at time t if and only if $f(t)$ is equal to $f_i(t)$ and

$$\forall_{j,k}[(t \leq t_j \leq t_k) \to f(t_k) \in R_i(f(t_j), t_k - t_j)].$$

A strategy profile $\vec{\sigma} = (\sigma_0, \sigma_1, \dots \sigma_n)$ assigns a single pure strategy to each agent. Since the game is deterministic, each strategy profile $\vec{\sigma}$ should produce a unique history $h(\vec{\sigma})$ at the end of the game. The expected value of profile $\vec{\sigma}$ is

$$E(\vec{\sigma}) = u(h(\vec{\sigma}))$$

where $u(h)$ is the size of the region guaranteed to contain the evader based on the pursuers' observation histories at the end of a game with history h. This value can be computed given the observation history $O_i(h)$ generated by history h,

$$u(h) = |\bigcap_{i=1}^{n} R_0^+(O_i(h), t_{end})|. \tag{2.4}$$

The utility for the pursuer team is $-E(\vec{\sigma})$, meaning the highest possible utility is zero, which happens when the evader is directly visible at the end of the game. We leave the objective function for the evader undefined, but set out to maximize the pursuers' utility $-E(\vec{\sigma})$ under a *worst-case* assumption: i.e. we assume that the evader will always select a path that minimizes the pursuers' utility.

2.3 Heuristic Strategy Generation

This section introduces the *limited Euclidean-space lookahead* (LEL) heuristic, a control action selection heuristic for pursuer agents in visibility-based pursuit-evasion games. LEL works by estimating how large the contamination region will be in the game's future if a pursuer agent follows a particular control action. As defined in Sec. 2.2, the contamination region is the set of locations where the evader could be located based on the information in an agent's observation history. The size of this region is proportional to the pursuer team's utility, defined in Sec. 2.2.4.

LEL is a continuous-space extension of the relaxed lookahead (RLA) heuristic introduced in prior work [35]. The RLA heuristic was designed to evaluate control actions for pursuer teams in gridworld domains. Paths generated using RLA were limited to movements between grid locations in one of four cardinal directions, and the algorithm required continuous, uninterrupted communication between agents. LEL overcomes these limitations by estimating the size of the contamination region using only an agent's observation history, as described in Sec. 2.3.1, and by using the Fast Marching Method (FMM) to compute the reachability and visibility information for each of the agents, described in Sec. 2.4.

Below, we provide a formal definition for the LEL heuristic in games where agents can move freely over Euclidean space and where communication between agents can be interrupted. This definition is described in terms of the formal definitions introduced in the previous sections. In Sec. 2.4, we provide a numerical approximation of LEL for two-dimensional spaces that can be computed efficiently.

2.3.1 Formal Definition of LEL

The LEL heuristic assigns preference values to individual states based on how the contamination region is expected to evolve over time. Correctly predicting how this region will grow is not possible without knowing the future movements of the pursuers and the evader. However, the LEL heuristic is able to estimate the size of this region by relaxing the problem space, disregarding individual paths, and instead evaluating all feasible paths that an agent may follow.

Given observation history O_i, the evader is guaranteed to be located somewhere in the region $R_0^+(O_i, t)$ at time t. Additionally, the region that can be observed by the pursuer team at time t is bounded by

$$V^+(O_i, t) = \bigcup_{j=1}^{n} V_j(R_j^+(O_i, t)) \tag{2.5}$$

which contains every location that a pursuer agent could observe at time t, given any path consistent with O_i. Using these bounds, agent a_i can approximate the region where the evader will be located at time t by computing

$$R_0^+(O_i, t) \setminus V^+(O_i, t) \subseteq \mathit{projected}_0(O_i, f_i, t)$$

which is the set of locations that the evader can reach by time t, minus the set of locations that may be visible to the pursuer team. This is a subset of the actual contamination region defined in Sec. 2.2.3. The value returned by the LEL heuristic is simply the weighted sum of this region's size over time,

$$U_{lel}(O_i, t) = \sum_{k=0}^{\infty} \gamma^k \left| R_0^+(O_i, t+k) \setminus V^+(O_i, t+k) \right| \tag{2.6}$$

where t is the current time and $\gamma \in [0, 1]$ is an exponential discount factor. Lower values for γ bias the heuristic towards the immediate future. In Sec. 2.4 we present an algorithm to efficiently compute a numerical approximation of LEL.

2.3.2 Action Selection

We can select control actions for pursuers agent by using the LEL heuristic to evaluate the state immediately produced by that action. The control actions space for agent a_i is a set $Q_i \subset \mathbb{R}^2$ of movement vectors where each movement vector $q \in Q_i$ has magnitude less than or equal to the maximum speed v_i of agent a_i,

$$Q_i = \{\langle q_x, q_y \rangle : q_x^2 + q_y^2 \leq v_i^2\}. \tag{2.7}$$

We can evaluate action $q \in Q_i$ by computing U_{lel} with the observation history that results from applying q to the current state. Let $f_i(t) + q$ be the location of pursuer a_i after applying action q for one time step. The value assigned to control action q is thus $U_{lel}(O_i \cup \langle a_i, \{f_i(t)+q\}, t+1\rangle, t+1)$, where $O_i \cup \langle a_i, \{f_i(t)+q\}, t+1\rangle$ is the modified observation history that accounts for the agent's movement. The preferred action for pursuer a_i is thus

$$q_i^*(O_i) = \arg\min_{q \in Q_i} U_{lel}(O_i \cup \langle a_i, \{f_i(t) + q\}, t + 1\rangle, t + 1). \tag{2.8}$$

which is the control action that minimizes the U_{lel} heuristic given the current observation history. In practice, this evaluation is performed for only a finite subset of the control actions in Q_i.

In our experiments in Sec. 2.5, a total of ten control actions per agent are evaluated: eight actions forming a uniform circle around the pursuer's current lo-

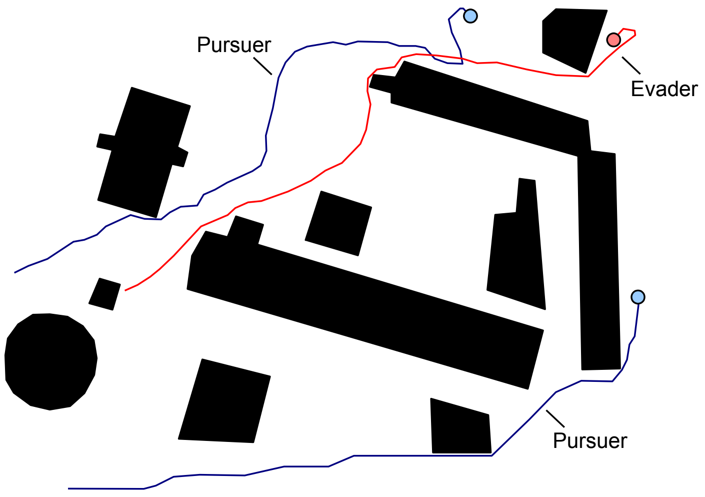


Figure 2.3: Paths generated using the LEL heuristic in a domain with two pursuer agents and one evader. The pursuer agents start in the lower left and move past obstacles while attempting to surround the evader from both sides.

cation, a single action to represent no movement, and a final action selected from a weighted average of the best two actions evaluated. The paths in Fig. 2.3 were generated using this technique.

If a tie occurs, the tie can be broken by re-computing LEL using the location of just one pursuer agent. To do this, substitute $V_i(R_i^+(O_i,t))$ for $V^+(O_i,t)$ in the heuristic and compute,

$$U_{tiebreaker}(O_i,t) = \sum_{k=0}^{\infty} \gamma^k \left| R_0^+(O_i,t+k) \setminus V_i(R_i^+(O_i,t+k)) \right| \tag{2.9}$$

which is equivalent to what the LEL heuristic would return if there were no other pursuer agents on the team. A tie can occur when some subset of the pursuers are the first to observe all of the locations that the evader can reach. In this case, the tie-breaker ensures that the remaining pursuers move into a reasonable position.

2.4 FMM-Based Implementation

This section introduces an efficient numerical approximation of the LEL heuristic using the Fast Marching Method [37]. The Fast-Marching Method (FMM) estimates the Euclidean shortest-path distances between a set of locations in a Cartesian grid. Given a sufficiently large grid, the values computed by the FMM form an arbitrarily close approximation of the actual distances. We use the FMM to compute the bounds of the contamination region, as well as the distances each agent must travel to reach locations in the domain.

Our algorithm is divided into two stages: first we compute reachability and visibility distances for each agent over a set of discrete grid points L_{grid}, then we combine that information to compute an evaluation function which approximates the value of U_{lel}. For this computation, we assume the agents have updated their observation history according to the rules described in Sec. 2.2.3.

Before calculating distances using the FMM, we must identify the set of locations in $L_{grid} \cap L_{free}$ over which to perform the calculation. If we assume that L_{free} is represented by a set of disjoint polygons in $\mathbb{R}^2$, then a linear-time discretization can be performed using point-in-polygon or similar algorithms [38]. The same technique can be used to discretize the contamination region or visibility regions.

Since multiple calls to the LEL heuristic will result in redundant work, improvements can be made by caching reachability information at critical locations, a process that is discussed in Sec. 2.4.3.

2.4.1 Path and Visibility Distance

The numerical approximation of LEL requires computing a set of reachability functions $\{rdist_{i,0}, rdist_{i,1}, \ldots rdist_{i,n}\}$ for agent a_i, where each function $rdist_{i,j}[l]$ returns the Euclidean shortest-path distance to location l from agent a_j's observed location at time t. If a_i does not know the exact location of agent a_j, then $rdist_{i,j}[l]$ is the shortest-path distance from $R_j^+(O_i, t)$, which is the set of locations guaranteed to contain a_j at time t based on a_i's observation history.

The values of $rdist_{i,j}[l]$ for all locations $l \in L_{grid}$ can be computed by the FMM in time $O(|L_{grid}|)$, where $L_{grid} \subset L_{free}$ is a discrete set of points in a Cartesian grid [39]. The FMM has the same complexity whether computing distances from a single location or set of locations, so it can be used even if the exact location of an agent is not known. The result of this computation is shown in Fig. 2.4b.

The numerical approximation algorithm also requires a set of visibility functions, $\{vdist_{i,1}, vdist_{i,2}, \ldots vdist_{i,n}\}$ where each $vdist_{i,j}[l]$ returns the shortest-path distance from agent a_j's location at time t to any location that can observe l. This can be defined in terms of $rdist_{i,j}$ as

$$vdist_{i,j}[l] = \min_{l' \in V_j(l)} rdist_{i,j}[l'] \tag{2.10}$$

where $V_j(l)$ is the set of locations visible to a_j from l.

Evaluating $vdist_{i,j}[l]$ is considerably more work than evaluating $rdist_{i,j}[l]$, since it requires computing the minimum distance over the locations in $V_j(l)$. Rather than

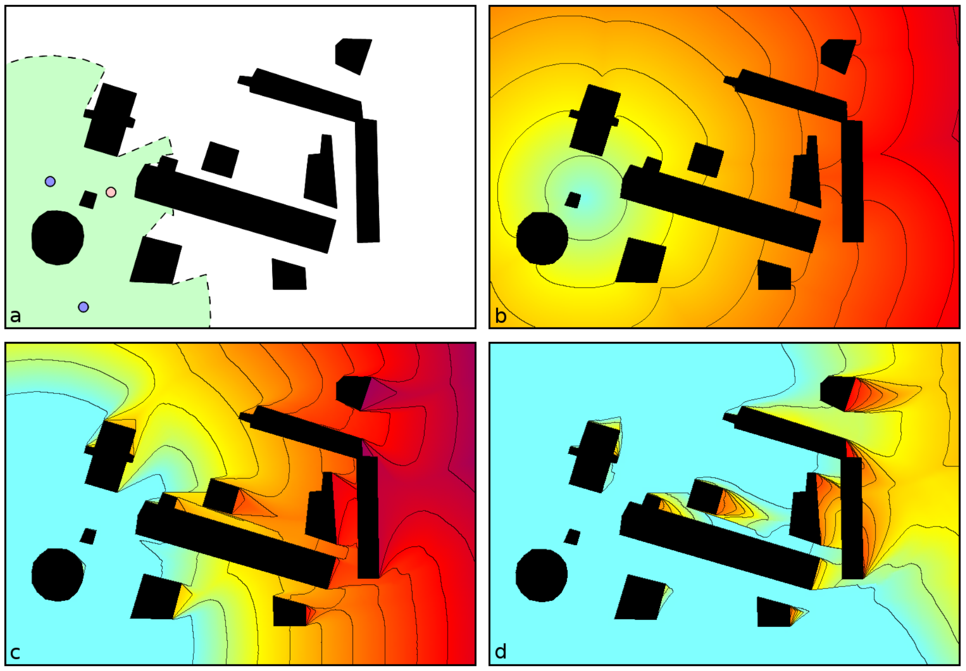


Figure 2.4: Steps in the computation of LEL: a) actual state of the game, b) reachability distance from the evader's location, c) visibility distance from the pursuers' locations, d) evaluation function used by the LEL heuristic.

computing this explicitly, we can approximate the visibility distance by computing $V_j(l)$ for a set of sample points L_{sample}, as shown in algorithm 1.

Algorithm 1 Compute agent a_i's visibility distance function $vdist_{i,j}$.

```
$L_{sample}$ = finite subset of $L_{free}$
for all $l \in L_{grid}$
    $vdist_{i,j}[l] \leftarrow \infty$
for all $l \in L_{sample}$
    for all $l' \in (L_{grid} \cap V_j(l))$
        $vdist_{i,j}[l'] \leftarrow \min(rdist_{i,j}[l], vdist_{i,j}[l'])$
```

The complexity of this algorithm is $O(k|L_{sample}|)$, where k is the average size of $L_{grid} \cap V_j(l)$. If each visible region $V_j(l)$ is represented as a polygon, we are able to compute algorithm 1 efficiently by taking advantage of scan-line rasterization techniques. An example of this computation is shown in Fig. 2.4c.

2.4.2 Numerical Computation of LEL

Given $\{vdist_{i,1}, vdist_{i,2}, \ldots vdist_{i,n}\}$ and $rdist_0$, we can calculate the difference in time between when a location can first be reached by the evader and when it can first be observed by one of the pursuers. This can be evaluated as follows

$$eval(l, O_i) = \min_j \left(g(v_j^{-1} \cdot vdist_{i,j}[l]) - g(v_0^{-1} \cdot rdist_{i,0}[l]) \right)$$

where $rdist_{i,j}[l]$ and $vdist_{i,j}[l]$ are the reachability and visibility distances of agent a_j to location l given history O_i. The value v_j is the maximum speed of agent a_j, and $g(v) = \int \gamma^v dv = \frac{\gamma^v}{\ln(\gamma)}$ is the indefinite integral of the exponential discount function. The numerical approximation of U_{lel} can be computed as

$$\tilde{U}_{lel}(O_i) = \frac{1}{|L_{grid}|} \sum_{l \in L_{grid}} \max(0, eval(l, O_i)) \tag{2.11}$$

which is the summation of $eval(l, O_i)$ over all the locations in L_{grid}, ignoring values less than zero. An example of this computation is shown in Fig. 2.4d.

To connect this algorithm to the formal definition of U_{lel} in equation 2.6, note that the size of an arbitrary region can be approximated by counting how many points in L_{grid} are contained by that region. The quality of the approximation depends on the size of the grid, but with any sufficiently large grid we can approximate the size of the region $R_0^+(O_i, t) \setminus V^+(O_i, t)$ to arbitrary precision and use that to estimate U_{lel}. However, rather than computing this region at each time step as is done in equation 2.6, we simply determine when each point in L_{grid} is first intersected by $R_0^+(O_i, t)$ and $V^+(O_i, t)$, then use the difference in time to determine how long the point was contained in $R_0^+(O_i, t) \setminus V^+(O_i, t)$. That is what is done in equation 2.11 using our algorithm, allowing us to leverage the Fast-Marching Method and avoid performing costly set operations over complex polygonal regions.

An example of the *rdist* and *vdist* value generated by this technique are shown in Fig. 2.4. The per-location evaluation of U_{lel} is represented as a heatmap, shown in the fourth image of the figure. Shaded areas represent locations where the evader is able to move before the pursuer can guarantee visibility, with darker areas indicating a greater time difference. If there are many such locations, then U_{lel} will return a high value, indicating that the state is poor for the pursuer team.

Most of the work performed evaluating a single control action can be re-used to evaluate multiple control action at once. Since the same observation history is used throughout this process, the reachability and visibility information for most of the agents does not need to be recomputed. Evaluating m possible control action for agent a_i involves computing $vdist_{i,j}$ and $rdist_{i,j}$ only once per agent $a_{j \neq i}$, while $vdist_{i,i}$ is computed m times.

2.4.3 Caching Optimizations

The running time of the numerical approximation algorithm can be reduced if the values of $rdist_{i,j}[l]$ and $vdist_{i,j}[l]$ are cached in advance. We can do this by caching these values critical points, i.e. locations in the domain that a large number of other paths are required to pass through. For a two-dimensional polygonal environment, the critical points are at the outward facing vertices of the obstacles, since the shortest path around an obstacle always passes through these vertices.

Let L_{vertex} be the set of vertices along the boundaries of the obstacles in the domain. Let $vcache_{i,j}[l, l']$ represent the cached visibility distance from some arbitrary location $l \in L_{vertex}$ to some arbitrary location l'. If agent a_j is located at $f_i(t)$, the value of $vdist_{i,j}[l]$ for all $l \in L_{grid}$ can be computed as follows:

Algorithm 2 Compute $vdist_{i,j}$ using critical point caching.

```
for all $l \in L_{grid}$
  $vdist_{i,j}[l] \leftarrow \infty$
for all $l \in (L_{sample} \cap V_j(f_j(t)))$
  for all $l' \in (L_{grid} \cap V_j(l))$
    $vdist_{i,j}[l'] \leftarrow \min(rdist_{i,j}[l], vdist_{i,j}[l'])$
for all $l \in (L_{vertex} \cap V_j(f_j(t))$
  $d = |l - l_{a_i}|$
  for all $l' \in L_{grid}$
    $vdist_{i,j}[l'] \leftarrow \min(d + vcache_{i,j}[l, l'], vdist_{i,j}[l])$
```

The advantage of using Alg. 2 is that it is no longer necessary to calculate $vdist_{i,j}[l]$ for any location l that isn't in the immediate line-of-sight of agent a_j. Instead, those distances can be determined based on the cached values in $vcache_{i,j}[l, l']$. As the agents move throughout the domain, the set $L_{vertex} \cap V_j(f_i(t))$ of vertices that are visible to agent a_i only change gradually as new vertices become visible and old ones become hidden. Thus, $vcache_{i,j}[l, l']$ can be computed once when a vertex first becomes visible, and re-used indefinitely until the cache becomes too large or the information is no longer needed.

It's best to use caching when the average number of obstacle vertices visible to an agent is much smaller than the number of points in L_{sample}. If the visibility polygons $V_j(l)$ are pre-computed for each point $l \in L_{sample}$, Alg. 2 has a computational complexity of $O((v + s)|L_{grid}|)$ where v is the average number of vertices visible to the agent, and s is the average number of points in $L_{sample} \cap V_j(l)$. If v and s are held constant, then the complexity is linear in the size of the grid. In our experiments, we compute the distances $vcache_{i,j}[l, l']$ in advance for all $l \in L_{vertex}$.

2.4.4 Complexity Analysis

The complexity of the approximation algorithm for the approximation algorithm introduced in this section is $O(n(V + R + |L_{grid}|))$, where n is the number of pursuers, R and V are the computational complexity of the algorithms used to generate $rdist$ and $vdist$. Once $rdist$ and $vdist$ are computed, computing LEL is simply a matter of computing the sum of the time differences between $rdist$ and $vdist$, which can be done in $O(n|L_{grid}|)$.

The Fast Marching Method can generate $rdist$ in linear time, $O(|L_{grid}|)$ [39]. The algorithm for computing $vdist$ has a complexity of $O((v+s)|L_{grid}|)$, as explained in Sec. 2.4.3. As a consequence, the time required to compute LEL increases only polynomially as the size of L_{grid} or the number of pursuers n increases.

The complexity of other important algorithms are as follows: computing $V_j(l)$ for any single location l can be done in $O(|L_{vertex}|)$, or linear in the number of vertices in the domain [40]. Similarly, the vertices visible from polygon L can be computed in $O(p \cdot |L_{vertex}|)$, where p is the number of vertices in the boundary of L. Both of these are used in the computation of $vdist$, and can be cached in advance.

2.5 Experimental Results

To evaluate the effectiveness of the LEL heuristic, we performed a series of experiments on randomly generated two-dimensional domains. For each experiment, we simulated games between a team of two or more pursuers and one evader. We evaluated the performance of the pursuer team against three different evader strategies: *evade pursuer* (S_{EP}), *evade visibility* (S_{EV}), and *reverse LEL* (S_{RL}). These evader strategies are defined in detail in Sec. 2.5.2.

For comparison purposes, we also evaluated the *min-distance* (MD) heuristic, an alternative control action selection heuristic for the pursuer team. The MD heuristic is a simple hand-coded rule that instructs the pursuers to follow the shortest-path to the evader. If the evader is not visible, pursuers using MD will follow the shortest-path to the closest possible location of the evader. This heuristic provides a baseline comparison for judging the quality of the strategies produced by LEL, and has been used for a similar purpose in the past [35].

Examples of the randomly generated domains used in these experiments are shown in Fig. 2.5. The free space in each domain was generated by producing a random spanning tree using a modified version of Kruskal's algorithm [41]. Additional random edges were added to the tree to increase connectivity. This technique produced domains with a variety of irregularly shaped obstacles, providing many opportunities for the evader to escape and hide.

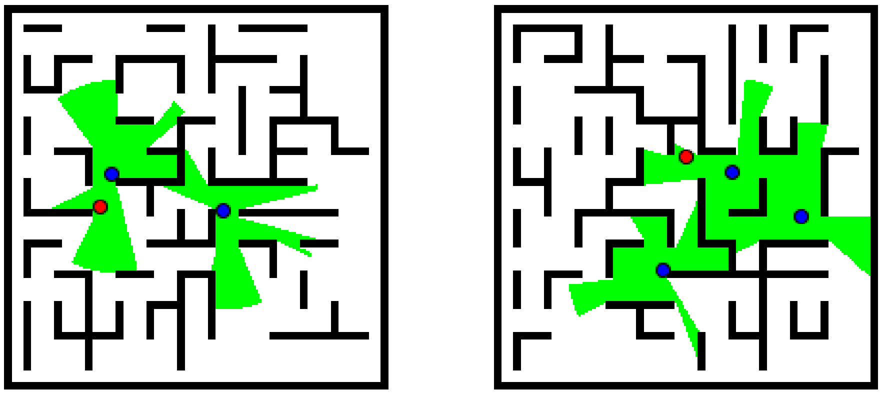


Figure 2.5: Two randomly generated domains with two and three pursuer teams.

2.5.1 Experimental Setup

For each experiment, the initial location of each pursuer was set to a random location in L_{free}, while the initial location of the evader was set to a random location in one of the pursuers' visible area. To make the game more challenging, the speed of the evader was set to 10% faster than the speed of the pursuers, meaning the evader was able to out-run the pursuers given enough time. Except where otherwise specified, the parameters used during the experiments are as follows: the size of $|L_{grid}| = 200 \times 200$, the sensor range $r_i = 50$ for all pursuers, the maximum speed $v_i = 1$ for all pursuers and $v_0 = 1.1$ for the evader, the U_{lel} discount factor $\gamma = 0.95$, and the duration of each game is 1000 time steps.

All results discussed in this chapter represent the average of 2000 randomly generated trials. For each trial, we used a different set of randomly generated obstacles and starting locations. The performance of each heuristic was measured by calculating the percentage of time the evader was visible to the pursuer and the average size of the contamination region throughout the trial.

2.5.2 Evader Strategies

To determine the behavior of the evader during each of the experiments, we specified three different evader strategies:

- *Evade pursuer* (S_{EP}) - evader selects a control action $q \in Q_0$ that maximizes its shortest-path distance to the nearest pursuer.
- *Evade visibility* (S_{EV}) - evader selects a control action $q \in Q_0$ that maximizes its shortest-path distance to the set of locations visible to pursuers.
- *Reverse LEL* (S_{RL}) - evader selects control action $q \in Q_0$ that maximizes U_{lel}, exactly opposite of the pursuers, attempting to maximize the projected size of the contamination region instead of minimizing it.

In addition to the behavior described above, both S_{EP} and S_{EV} exhibit a special "escape" behavior when the evader is visible to pursuers and the evader's distance to the nearest pursuer is less than d_{min}. In this situation, the evader will travel to the nearest location that is at least d_{min} away from the pursuers. This is done to avoid situations where the evader would otherwise become trapped in a corner and stop moving. This problem does not occur with the S_{RL} strategy, so the escape behavior is not triggered when S_{RL} is used. The value of d_{min} was set to 20 for all experiments discussed in this chapter.

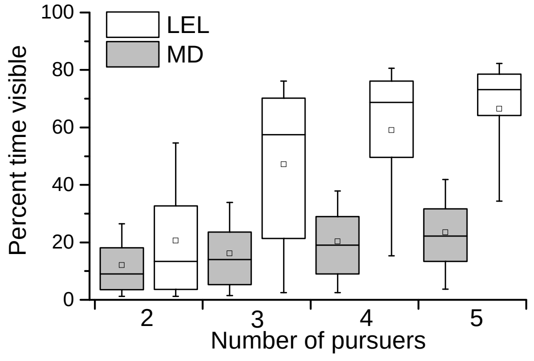
The shortest path distances for S_{EP} and S_{EV} can be computed using the Fast Marching Method, as described in section Sec. 2.4.1. For each evader strategy, we assume the evader has complete knowledge of the state of the world, including the current locations of the pursuers.

2.5.3 Pursuer Performance

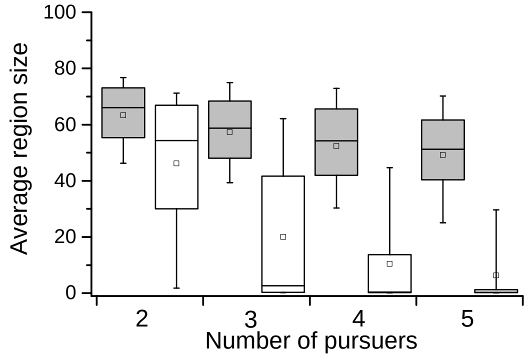
The first set of experiments were designed to measure the performance of the LEL and MD heuristic in games where the number of pursuers varied between two and five agents. Performance was measured against each of the evader strategies introduced in Sec. 2.5.2. Simulations were run for 1000 time steps and the average size of the contamination region and percentage of time the evader was visible were recorded. We assumed that pursuers were in constant communication during these experiments, so no communication interruption was introduced.

The results in Fig. 2.6 show that teams using the LEL heuristic were significantly more effective at pursuing the evader compared to teams using the MD heuristic, and that the difference in performance grew as the size of the team increased. Teams using LEL had visibility on the evader for a higher percentage of time, and the average size of the contamination region was smaller.

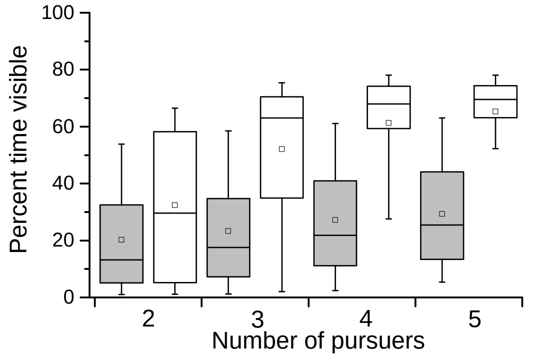
When comparing the percentage of time a team of two pursuers had visibility on evaders that were using the S_{RL} strategy, there was a 1.7 times difference between the LEL and MD heuristic. For teams of three pursuers, the difference between LEL and MD grew to 2.9 times. A similar relationship between LEL and MD exists when comparing the average size of the contamination region. For these experiments, the benefit of using LEL was apparent against all three evader strategies, with LEL



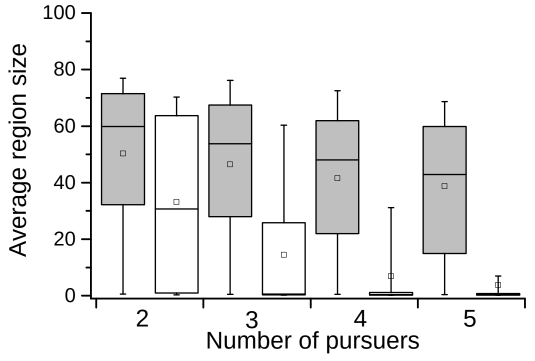
(a) Percent time visible vs. S_{RL} evader



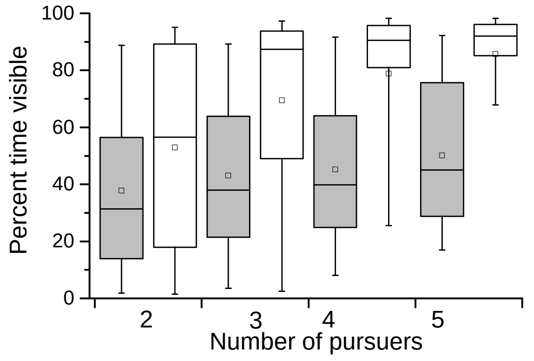
(b) Contamination region size vs. S_{RL} evader



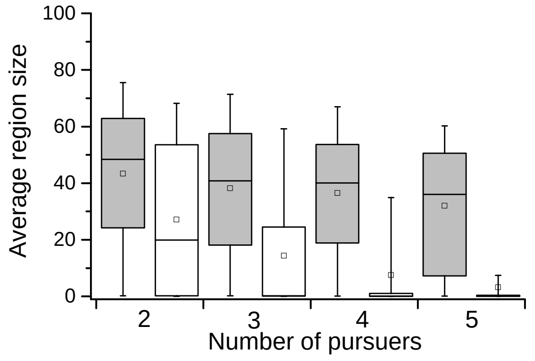
(c) Percent time visible vs. S_{EV} evader



(d) Contamination region size vs. S_{EV} evader



(e) Percent time visible vs. S_{EP} evader



(f) Contamination region size vs. S_{EP} evader

Figure 2.6: Comparison of LEL and MD heuristics with different teams sizes against the S_{RL}, S_{EV} and S_{EP} evader strategies: (left) the percentage of time the evader was visible, higher is better (right) the average size of the contamination region, lower is better. The box plots show the median, upper and lower quartile. The whiskers show the upper and lower decile. The mean value is represented by a square.

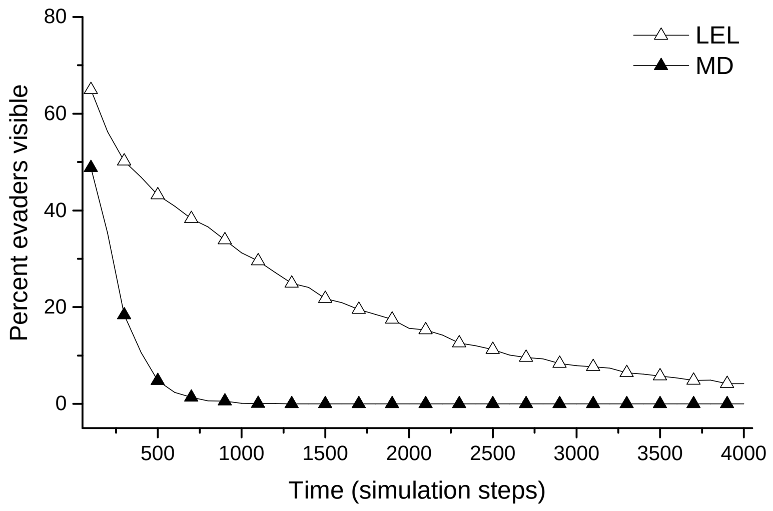


Figure 2.7: Percentage of games where evader using the S_{RL} strategy are visible to a team of two pursuers using LEL and MD heuristics as the simulation time increases.

outperforming MD in every case.

Of the three evader strategies, S_{RL} proved the most challenging for pursuers, since it resulted in the lowest performance for pursuers regardless of the size of the pursuer team or the heuristic used. This is consistent with the idea that LEL is an effective heuristic, since S_{RL} is simply the negation of LEL applied to the evader. Between the other two evader strategies, S_{EV} was slightly more challenging than S_{EP}, which makes intuitive sense because S_{EV} explicitly avoids the area visible to pursuers, while S_{EP} does not.

The results in Fig. 2.7 show that, as the game progresses, fewer and fewer evaders remain visible to the pursuer team. The LEL heuristic is significantly better than MD at maintaining visibility on the evader over time. The consistent decline in both cases can be attributed to the fact that the evader travels faster than the pursuer, meaning the evader is highly likely to break line-of-sight eventually. The LEL heuristic is simply better at deferring that outcome until later.

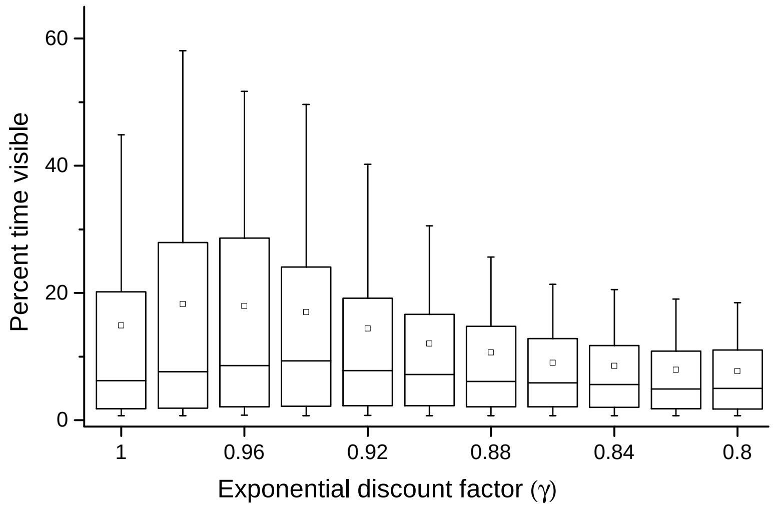


Figure 2.8: Percentage of time an evader using the S_{RL} strategy is visible to a team of two pursuers using LEL as the exponential discount factor γ is varied. Recall that γ is used in the computation of U_{lel}, as described in Sec. 2.3.1

The effectiveness of the LEL heuristic is influenced by the choice of exponential discount factor $\gamma \in [0, 1]$. Recall from Sec. 2.3.1 that γ is used to bias the computation of U_{lel} towards near-term predictions about the size of the contamination region. Lower values for γ mean the heuristic will weigh the immediate future more heavily. When $\gamma = 1$, both the near and long term are weighted equally. The reason for introducing any bias at all, is that predictions about the near future are more likely to be accurate, and provide a better estimate of the expected utility.

The results in Fig. 2.8 show how the amount of time the evader is visible changes as the value of γ is varied. For these experiments, we ensured that changes in γ only affected the LEL heuristic used by pursuers. The strategy used by the evader was S_{RL} with $\gamma = 0.95$, and its parameters were held constant. The experimental setup was otherwise the same as described previously.

The best performance was obtained when γ was between 0.98 and 0.94, cor-

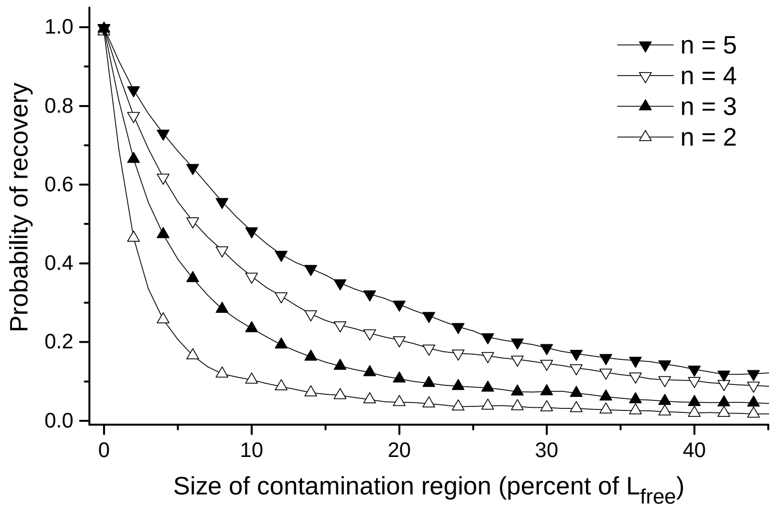


Figure 2.9: Probability of recovering an evader using the S_{RL} strategy within 500 time steps given on the current size of the contamination region. Results are shown for pursuer teams of size n using the LEL heuristic.

responding to a 2% to 6% reduction in importance weight per time step. The best performance is achieved when $\gamma < 1$, because long-term predictions are less accurate, so they should not be weighed as heavily. At the opposite extreme, when γ approaches 0, the pursuers become concerned only with the present, failing to take advantage of the predictive capabilities of the heuristic. Similar results were obtained when using S_{EV} and S_{EP} evader strategies.

It is important to remember that games do not end when an evader breaks line-of-sight. The pursuer team has the ability to recover the evader by reasoning about the boundaries of the contamination region, something that is incorporated into the design of LEL. The results in Fig. 2.9 show the probability of a pursuer team regaining visibility on an S_{RL} evader at some point in the next 500 time steps given the current size of the contamination region. These results were gathered from an expanded set of trials using the same setup as Fig. 2.6.

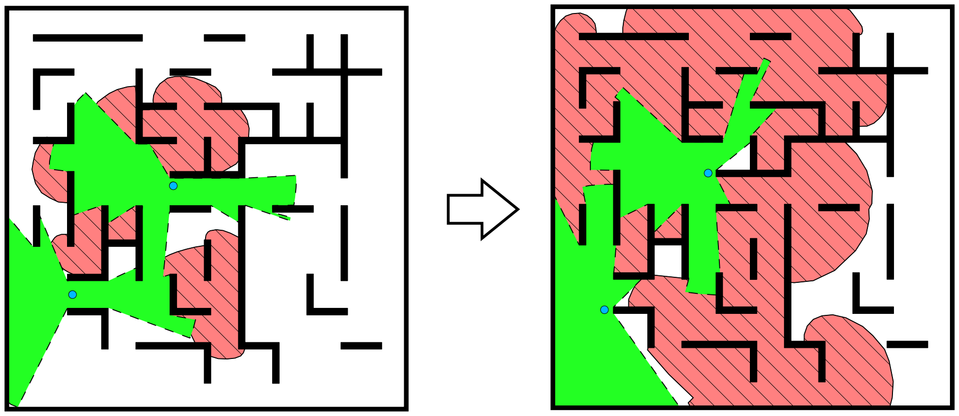


Figure 2.10: An evader escaping detection in a randomly generated game. The pursuers are unable to completely contain the growth of the contamination region, leading to poor performance from the LEL heuristic.

When the size of the contamination region is small (less than 5% of the total area in L_{free}) then there is 20% likelihood that the evader will be recovered by a team of two pursuers, and a 68% likelihood the evader will be recovered by a team of five pursuers in the next 500 time steps. However, there is an inverse relationship between the size of the contamination region and the likelihood of recovery, with the likelihood of recovery dropping to less than 5% for a team of two pursuers when the contamination region exceeds 20% of the total area.

The above results suggest that, although pursuers using LEL have the ability to recover visibility on lost evaders, they do poorly when uncertainty about the evader's location is very high. From a qualitative perspective, pursuers using LEL have the tendency to stall or stop moving when that are no control actions that lead to a near-term reduction in the size of the contamination region. This is apparent in Fig. 2.10, where the rapidly expanding contamination region can no longer be contained by the pursuers. The inability of LEL to effectively recover from these situations motivates the probabilistic extension discussed in Chapter 3.

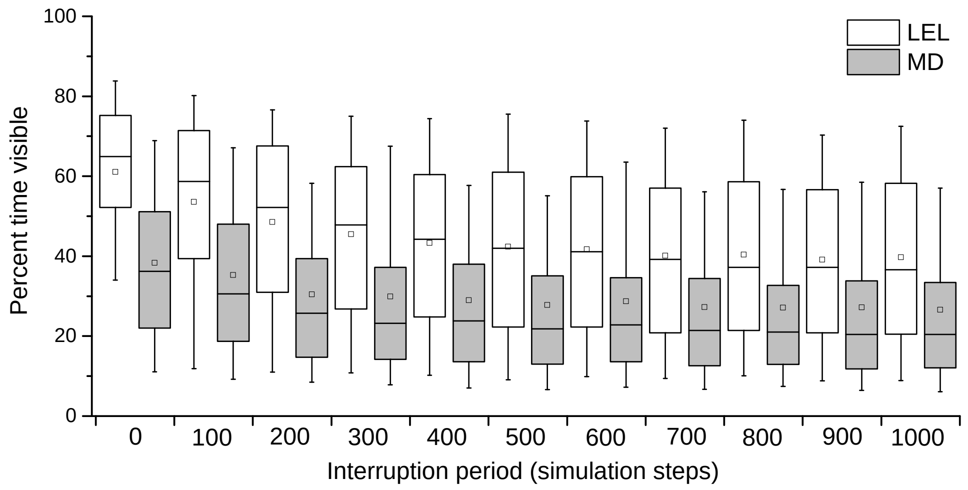


Figure 2.11: Results for LEL and MD when communication is interrupted. As the time between updates increases, agents are able to communicate less frequently.

2.5.4 Interrupted Communication

The second set of experiments were designed to measure the performance of the LEL and MD heuristics when communication between pursuers was interrupted. Recall from Sec. 2.2.3 that the observation history of a single pursuer is updated only periodically with information from other pursuers. Between these periodic updates, an individual pursuer must continue to generate control actions based only on the most recent knowledge of other pursuers' location and observations. The LEL heuristic has been designed to deal with communication interruption by making projections about the location of the other pursuers and their observations in absence of up-to-date information.

For these experiments, we set a fixed interruption period during which no communication between agents was allowed. If the interruption period was set to 100 simulation steps, then agents could communicate only once every 100 steps and not any time between. When communication did occur, agents exchanged all the information in their current observation history.

Fig. 2.11 shows the effect of varying the interruption period for a team of two pursuers versus an evader using the S_{RL} strategy. As expected, when the interruption period was increased, the pursuers became less successful at pursuing the evader. However, the pursuers performed better when using the LEL heuristic, even when the interruption period was very long. For pursuers using LEL, the average percentage of time the evader was visible dropped from 61% to 39% as the interruption period increased from 0 to 1000. For pursuers using MD, the average dropped from 38% to 26% as the interruption period increased.

To put these results into perspective, when the interruption period was set to 1000 time steps, agents were only able to communicate once throughout the entire game. Pursuers with a high level of interruption performed better using LEL than pursuers with no interruption using MD. The performance of pursuers declined as interruption increased, but the decline was not especially dramatic, indicating that the heuristic is at least partly resilient to these interruptions.

2.5.5 Running-Time

The final set of experiments were designed to evaluate the running time and computational complexity of the algorithm. The results in Fig. 2.12 show the average CPU time[1] for a single agent to select a control action using the LEL heuristic as

[1]All experiments were performed using a single core 2.40 GHz Intel Xeon processor running Java Virtual Machine 6.

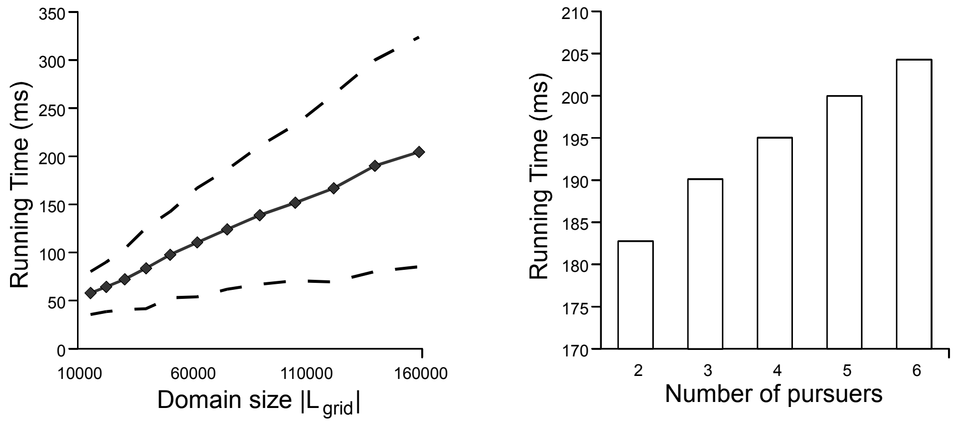


Figure 2.12: (left) Running time in milliseconds for an agent to select its next move using the LEL heuristic. The dashed lines show one standard deviation from the mean. (right) Running time for LEL using a 400x400 grid given different team sizes.

the size of the domain and number of pursuers are varied. In these experiments, the U_{lel} heuristic was evaluated nine times per decision and once per control action as discussed in Sec. 2.3.2.

The relationship between CPU time and the size of the domain (both in number of obstacles and the size of L_{grid}) was approximately linear. The relationship between team size and average CPU time per agent was also approximately linear. For the largest games we evaluated, with six pursuer agents and an L_{grid} size of 200×200, the average decision time was under half a second. Since the algorithm has a low running time, and the runtime complexity is approximately linear, this approach is well scalable to larger problems. This is a notable difference from approaches such as LaValle et al. which scale exponentially as the size of the domain or number of agents increases [7].

2.6 Discussion

This chapter introduced the *limited Euclidean lookahead* (LEL) heuristic, a control action selection heuristic for teams of pursuer agents in visibility-based pursuit-evasion games. The LEL heuristic makes predictions about the future set of locations that may contain the evader (the contamination region) by forward simulating relaxed version of the game. Experimental results show that pursuer teams using LEL are better at maintaining visibility on an evasive target than pursuer teams using the MD heuristic, which follows the shortest path to the target's current location. This same result was demonstrated against three different evasion strategies. The LEL heuristic is also resilient against communication interruptions between agents, maintaining its superiority over the MD heuristic even when communication is severely reduced. Finally, the LEL heuristic can be computed efficiently, scaling linearly with the size of the domain and number of agents, suggesting that the heuristic can be used in time and resource constrained environments.

Agents that use the LEL heuristic exhibit very different behavior from pursuers that simply follow the shortest path to the evader. Using LEL, typically one pursuer will follow the evader closely, while the remaining agents position themselves around the domain in a way that will corner the evader. An example of this behavior is shown in Fig. 2.3, where the upper-most pursuer follows the evader directly, while the lower pursuer moves along the edge of the domain to intercept the evader as it passes behind the obstacles. This "division of labor" is often seen when LEL is used, even though each pursuer selects its own control actions independently.

2.6.1 Generalizations of LEL

Although the LEL heuristic introduced in this chapter is defined in terms of two-dimensional Euclidean space, it is straightforward to generalize the approach to higher-dimensional spaces. The formalism in Sec. 2.2 can be relaxed to $L_{free} \subseteq \mathbb{R}^N$, and corresponding changes to the reachability and visibility functions R and V can be made which accommodate higher dimensionality.

Higher dimensional implementations of the Fast Marching Method already exist [42, 43], and their incorporation into the reachability function R would require little to no change in how the heuristic is formulated. For configuration spaces where agents are subject to differential motion constraints, Fast Marching Trees can be used as an alternative to the Fast Marching Method [43]. Additionally, the computation of V for dimensions $N > 2$ can be performed via ray-tracing. Although ray-tracing is significantly slower than the two-dimensional algorithm cited in Sec. 2.4.4, it can be easily parallelized if performance requirements demand it [44].

The value of LEL as a heuristic comes largely from its ability to efficiently redistribute agents around the environment. It is possible to imagine LEL being used to evaluate alternate objective functions and not just visibility loss. As an example, the objective function in Sec. 2.4.2 could be modified to distribute agents such that the expected wait time to observe a set of target locations is locally minimized. The general characteristic of objective functions for LEL is that they accept a set of the distance functions as input (e.g. $rdist$ and $vdist$), and return a single utility value. Any additional input features (e.g. potential fields marking distance to landmarks) are likely to be domain-specific, but could reuse many of the same algorithmic tools such as the Fast Marching Method and visibility sampling.

2.6.2 Limitations and Future Work

One clear limitation of the LEL heuristic is that there is a very low likelihood a lost evader will be recovered once the size of the contamination region is sufficiently large. This scenario is depicted visually in Fig. 2.10, where the evader goes undetected by pursuers and the contamination region fills the majority of the environment. Since the LEL heuristic reasons only about the bounds of the contamination region, and not the probability of the evader being located anywhere within those bounds, situations such as these result in poor behavior from the heuristic.

As explained in Sec. 2.1.2, due to the work of LaValle et al. we know that for a large class of problems there is no sequence of actions that can be performed which is guaranteed to detect the evader [2, 7]. This is a fundamental limitation of any algorithm that assumes a worst-case model of the opponent, including the LEL heuristic, as well as the algorithm introduced by LaValle et al. [7]. Rather than making a best-effort attempt to recover the evader, many of these approaches simply fail to produce a solution, or as is the case with LEL, fall into a state of inaction. In the next chapter, I demonstrate how this limitation can be addressed by incorporating a probabilistic model of the opponent into the algorithm. I also provide some discussion about what practical challenges must be overcome before implementing the LEL heuristic on a real-world robotic system.

Chapter 3: Pursuit-Evasion with Probabilistic Opponent Models

One major limitation of the LEL heuristic identified in Chapter 2 is that pursuers using the LEL heuristic have difficulty regaining visibility on the evader once the contamination region becomes sufficiently large. To address this limitation, this chapter introduces the *probabilistic lookahead* (PLA) heuristic, a control action selection heuristic for the pursuer team that incorporates a probabilistic model of the evader. The main benefit of PLA over LEL is that it allows the pursuer to search for the evader based on where it is likely to be located, and is not restricted to reasoning only about the bounds of the contamination region.

To support the PLA heuristic, this chapter introduces several probabilistic opponent models backed by a particle filtering technique. Sec. 3.1.3 demonstrates how the distribution of particles in the filter can be influenced by a user-defined potential field, while Sec. 3.1.4 shows how these potential fields can be learned by collecting historical data from interaction traces of previous games.

I show experimentally that the PLA heuristic is able to outperform the LEL heuristic in randomly generated pursuit-evasion games. I also demonstrate that the PLA heuristic achieves superior recovery times compared to the algorithm introduced by LaValle et al. [7] in two case studies on example domains.

3.1 Opponent Modeling

This section provides an overview of how particle filtering techniques can be combined with weighted potential fields to produce different probabilistic models of the evader. These models include purely random opponent models (Sec. 3.1.2), opponent models based on weighted potential fields (Sec. 3.1.3), and opponent models learned from historical data (Sec. 3.1.4). Instructions on how integrate these opponent models into the PLA heuristic is provided in Sec. 3.2.

3.1.1 Particle Filtering

Particle filtering, also known as Sequential Monte Carlo, is a sampling-based probability density estimation technique for systems with observable and hidden variables. Alternative hypotheses about the state of the system are represented as individual particles, which collectively form an estimate of the posterior probability distribution of the system. Particle filters have appeared in robotics literature as an alternative to Kalman Filters for estimating the state of non-linear, non-gaussian systems [45, 46]. Well-known applications of this technique include FastSLAM [47] and Monte-Carlo Localization [48].

Particle filtering is directly applicable to the visibility-based pursuit-evasion problem since each particle can represent an alternative hypothesis about the evader's location while it is hidden from view. The movement of particles can be sampled directly from a probabilistic motion model of the evader. Since each particle can be simulated individually, there are very few constraints on the type of model that can be used. The weights assigned to each particle can be updated given the pursuers' sensor model and observation history.

In this paper, particle system X_t is a set of states $\{x_{0,t}, x_{1,t}, \ldots x_{n,t}\}$ representing possible locations of the evader at time t. The location each particle is updated by performing Monte Carlo sampling over some opponent model M, where $M(x_{i,t+k}|x_{i,t}, O_t)$ defines the probability that an evader will transition from state $x_{i,t}$ to state $x_{i,t+k}$ given observation history O_t. Since the purpose of the particle filter is to model an unseen evader, the posterior probability of state $x_{i,t}$ is reduced any time that $x_{i,t}$ intersects the visible area of the pursuer agents. Thus, the probability assigned to state $x_{i,t+k}$ given state $x_{i,t}$ and O_t is,

$$P(x_{i,t+k}|x_{i,t}, O_t) = (1 - P_{obs}(x_{i,t}|O_t))M(x_{i,t+k}|x_{i,t}, O_t), \tag{3.1}$$

where $P_{obs}(x_{i,t}|O_t)$ is the probability that the pursuer team will detect an evader at state $x_{i,t}$ at time t. In practice, the value of $P_{obs}(x_{i,t}|O_t)$ will always be either 1, indicating that $x_{i,t}$ is in the pursuers' sensor range, or 0 otherwise.

The posterior probability $P(x_{i,t}|O_t)$ of particle i reaching state $x_{i,t}$ from its initial state $x_{i,0}$ can be derived recursively using the formula

$$P(x_{i,t}|O_t) = P(x_{i,t}|x_{i,t-k}, O_t)P(x_{i,t-k}|O_{t-k}), \tag{3.2}$$

which terminates trivially at the base case $P(x_{i,0}|O_0) = P(x_{i,0})$ where $P(x_{i,0})$ is the prior probability of state $x_{i,0}$. Prior probabilities are assigned when the particle system is initialized, which is usually when visibility of the evader is initially lost.

Each particle is assigned an importance weight $w_{i,t}$ meant to approximate the relative posterior probability $P(x_{i,t}|O_t)$. The weights of all particles are initially set to $1/|X_0|$, meaning that they are all equally likely. During each time step, the weights are updated such that $w_{i,t+k} = w_{i,t}P(x_{i,t+k}|x_{i,t}, O_t)$. If the weight $w_{i,t}$ of any particle $x_{i,t}$ drops below a critical value w_{floor}, then particle $x_{i,t}$ is discarded and the weights of the remaining particles are normalized.

If particle $x_{i,t}$ is discarded, a replacement particle is resampled immediately by "splitting" one of the existing particles in X_t. This creates two particles at the same location with half the weight of the original. The particle that is split is selected with a probability proportional to its weight. If at any point there are no remaining particles in X_t, new particles are sampled uniformly from the set of possible locations for the evader.

3.1.2 Randomized Models

One very simple approach to modeling an opponent is to treat the opponent as if they are moving randomly. Although this modeling assumption may seem unrealistic, prior research has demonstrated its effectiveness as a heuristic in adversarial games with high levels of uncertainty [49]. Randomized models also have the advantage of being computationally efficient and requiring little to no prior knowledge about the evader's actual behavior.

To implement a randomized opponent model, it is necessary to define what

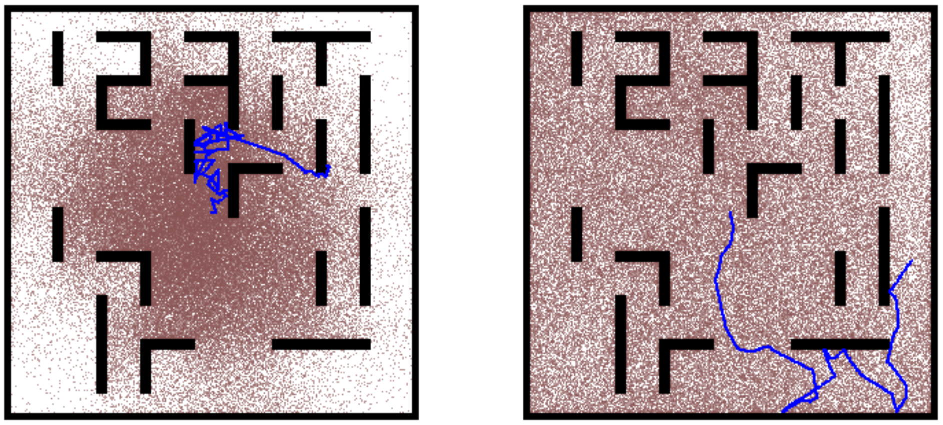


Figure 3.1: Particle dispersion using the *random walk* M_{RW} (left) and *persistent random walk* M_{PRW} (right) opponent models with $p_{change} = 0.1$ and $\sigma_{change} = \pi/6$. All particles were initialized at the center point. The path followed by a single particle is marked with a blue line.

"random" means in the context of movement in Euclidean space. According to the definition of reachability in Sec. 2.2, an evader can transition from its initial location l_0 to some location in $R_0(l_0, k)$ in time k. However, many possible distributions over this set of locations could be considered "random." In this paper, I have selected two such alternatives to describe a random opponent:

- *Random walk* (M_{RW}) - each particle $x_{j,t}$ follows control action $q_{j,t}$ which changes with probability p_{change}, where $q_{j,t}$ is sampled uniformly from the set of all possible control actions.

- *Persistent random walk* (M_{PRW}) - same as M_{RW}, except action $q_{j,t}$ is selected by rotating particle $x_{j,t}$'s previous control action $q_{j,t-k}$ by some angle ψ_X sampled from a Gaussian distribution with standard deviation σ_{change}.

The primary difference between these models is the rate of dispersion and shape of the paths they produce. Particles using M_{RW} disperse more slowly and exhibit paths resembling Brownian motion. Particles using M_{PRW} disperse more quickly and have longer periods of stable movement. Decreasing the values of p_{change} or σ_{change} results in more linear paths, since control actions will not change as frequently or significantly. Example paths for these two models are exhibited in Fig. 3.1 along with the resulting particle distribution.

To prevent particles from getting stuck on walls, both M_{RW} and M_{PRW} will only select control actions that face away from obstacles. If a particle's current action would result in a collision, the control action is resampled immediately, rather than waiting for a collision to occur. Otherwise, control actions are sampled randomly at each time step with probability p_{change}.

3.1.3 Weighted Potential Fields

One alternative to a purely random opponent model is to guide the motion of particles using a potential field. Potential fields specify a gradient over the state space such that goal locations can be reached via local hill-climbing. Potential fields have previously appeared in robotics literature as an efficient way to perform motion planning and obstacle avoidance [50]. They are particularly useful for particle systems, since a single potential field can guide the motion of every particle, incurring little additional cost as the number of particles increases.

If F is a potential field, then the potential at state $x_{j,t}$ is given by $F(x_{j,t})$. This potential value can be thought of as the evader's preference for state $x_{j,t}$. When applied to a particle filter, particles are locally attracted to states that have high potential and repelled from states with low potential. Since hill-climbing is used, particles may become concentrated in areas that represent local maxima.

A particle's control action $q_{j,t}$ given F is determined by finding the gradient vector $\nabla F(x_{j,t})$ that maximizes potential at $x_{j,t}$. To add noise to the system, action $q_{j,t}$ is subsequently rotated some randomly selected angle ψ_X. As with M_{PRW}, the value of angle ψ_X is selected from a Gaussian distribution with standard deviation of σ_{change}. Each particle's control action is resampled with probability p_{change} per time step. This results in a set of particles X which closely follow the gradient of the potential field, but with enough variation to generate multiple hypotheses.

Using potential fields, it is possible to generate matching opponent models for the S_{EP} and S_{EV} evader strategies introduced in Sec. 2.5.2.

- *Evade pursuer* (M_{EP}) - each particle $x_{j,t}$ follows a control action $q_{j,t}$ that maximizes its shortest-path distance to the nearest pursuer.

- *Evade visibility* (M_{EV}) - each particle $x_{j,t}$ follows a control action $q_{j,t}$ that maximizes it shortest-path distance to the set of locations visible to pursuers.

Potential fields for M_{EV} and M_{EP} can be generated in $O(|L_{grid}|)$ time using the Fast-Marching Method, as mentioned in Sec. 2.5.2. An example of the potential field for M_{EV} is shown in Fig. 3.2 along with the resulting particle distribution. The behaviors of both M_{EV} and M_{EP} are similar when the evader is far away from the pursuers, since the shortest path to the nearest pursuer and shortest path to the pursuers' visible region are approximately the same. Noise is added to both of these models by specifying some $\sigma_{change} > 0$.

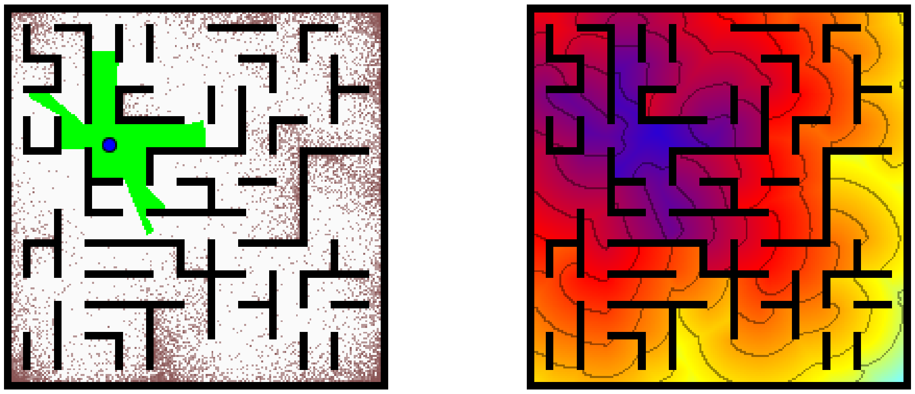


Figure 3.2: Example particle system (left) and potential field (right) when using the *evade visibility* (M_{EV}) opponent model. Particles flow along the gradient in the direction of increasing potential, denoted by brighter colors.

Generating an opponent model for S_{RL} is not as simple as generating models for S_{EP} or S_{EV}, since a potential field for S_{RL} cannot be efficiently computed given only a description of the strategy. However, in the following section, we demonstrate how the motion preferences of more complicated strategies such as S_{RL} can be encoded in potential fields generated from historical training data.

3.1.4 Learning from Example Games

The opponent models described in the previous sections assume that either no information about the evader's strategy is known, or that the evader's strategy can be defined explicitly in terms of a potential field. However, there may be cases where the only knowledge about an evader's strategy comes through prior observations of the evader's movement. This section demonstrates how the motion preferences of such an evader can be encoded in an opponent model by generating approximate potential fields from historical training data.

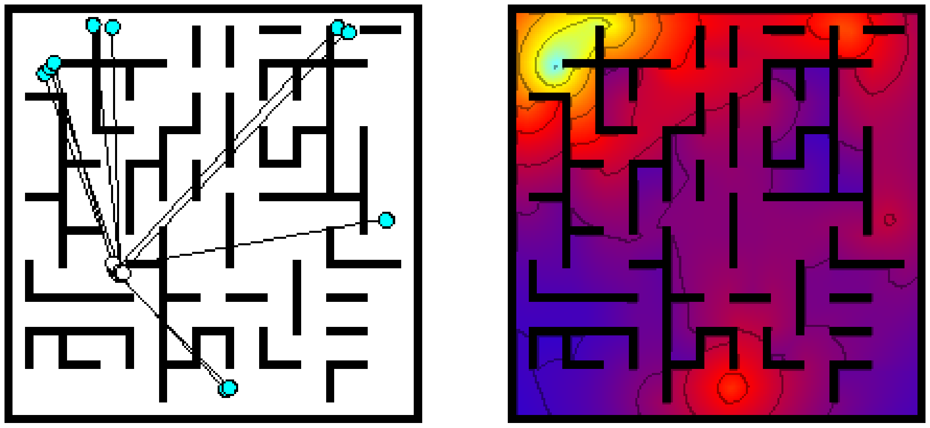


Figure 3.3: Location pairs drawn from historical data (left) and corresponding potential field (right) using k-nearest estimation ($k = 10$). Historical data is sampled from 100 games between one pursuer and one evader. Locations of the pursuer are shown in white, while locations of the evader are shown in cyan.

The conditional potential field $F_H(x_i|Y_j)$ defines the potential at state x_i given input vector $Y_j = \{y_{j,1}, y_{j,2}, \ldots y_{j,n}\}$, where each $y_{j,k} \in Y_j$ represents the current location of pursuer a_k. The purpose of F_H is to encode the motion preference of the evader as a function of the pursuers' locations. $F_H(x_i|Y_j)$ is computed using a set of historical data H, where each location pair $\langle x, Y \rangle \in H$ is sampled from interaction traces of previous games.

Any historical data set is unlikely to exhaustively cover the state space, so it is necessary to generalize from the data that is available. When computing F_H, the potential at any given state is defined by performing a weighted average over the historically observed states. If $H = \{\langle x_0, Y_0 \rangle, \langle x_1, Y_1 \rangle, \ldots \langle x_m, Y_m \rangle\}$ is the set of historical location pairs, then potential $F_H(x_i|Y_j)$ is given by the formula

$$F_H(x_i|Y_j) = \frac{\sum_{k=1}^{m} e^{-\lambda_x d(x_i, x_k)} e^{-\lambda_y D(Y_j, Y_k)}}{\sum_{k=1}^{m} e^{-\lambda_y D(Y_j, Y_k)}} \tag{3.3}$$

where $\lambda_x, \lambda_y \in [0, \infty]$ are exponential decay constants, $d(x_i, x_k)$ is the shortest-path distance between states x_i and x_k, and $D(Y_j, Y_k)$ is an equivalent shortest-path distance metric for the paired state vectors Y_j and Y_k, defined by

$$D(Y_i, Y_j) = \sqrt{\textstyle\sum_{k=1}^{n} d(y_{i,k}, y_{j,k})^2}. \tag{3.4}$$

The intuition behind Eqn. 3.3 is that samples in H where the pursuers are located nearer to the state vector Y_j should have greater influence on the resulting potential field. The term $e^{-\lambda_y D(Y_j, Y_k)}$ can be thought of as the weight for the kth sample in H, such that $F_H(x_i|Y_j)$ computes a weighted average of $e^{-\lambda_x d(x_i, x_k)}$ over all samples k. The reason for incorporating exponential decay into the formula is to impose a falloff as the actual state diverges from the sample data.

Due to the exponential falloff, samples in H that are sufficiently far away from state vector Y_j have very little influence on the value of $F_H(x_i|Y_j)$. For this reason, it is reasonable to restrict the computation of F_H to only samples that are the k-nearest neighbors of Y_j. The result of this is depicted in Fig. 3.3, where the potential field is generated from only the ten nearest samples.

The *learned preference* opponent model (M_{LP}) is generated by using potential field F_H to guide a particle filter. This model is depicted in Fig. 3.4 for a single pursuer and evader. The historical preferences of the evader are encoded in F_H, while the feasibility of individual hypotheses is determined by the flow of particles. Together they form an estimate of the probability density over the evader's possible locations. The viability of this model is evaluated in Sec. 3.3.

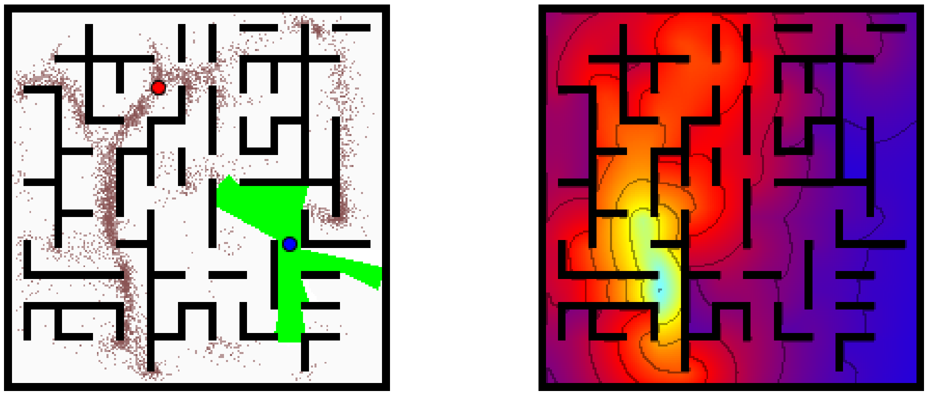


Figure 3.4: Example particle system (left) and learned potential field (right) when using the *learned preference* (M_{LP}) opponent model in an example game between one pursuer and one evader. Particles flow along the gradient in the direction of increasing potential, denoted by brighter colors.

3.2 Probabilistic Lookahead Heuristic

Now that we have a set of opponent models for hypothesizing about the evader's location, it is necessary to describe how those opponent models can be integrated into a planning heuristic for the pursuers. This section introduces *probabilistic lookahead* (PLA), a control action selection heuristic for the pursuer team which relies on the particle filtering methods described in this chapter.

PLA's evaluation function U_{pla} can be used to select control actions in a manner similar to U_{lel} as described in Sec. 2.3.2. The key difference between U_{lel} and U_{pla} is that instead of estimating the future size of contamination region, U_{pla} estimates the probability of observing the future location of each particle in X_t. The formula for U_{pla} is given by,

$$U_{pla}(X_t, O_t) = \sum_{k=0}^{d} \gamma_{pla}^{k} \frac{\sum_{i=0}^{n} P(x_{i,t+k}|O_t) obs(x_{i,t+k}, k, O_t)}{\sum_{i=0}^{n} P(x_{i,t+k}|O_t)} \tag{3.5}$$

where X_t is the particle system representing locations of the evader, $P(x_{i,t+k}|O_t)$ is the estimated probability of future state $x_{i,t+k}$ given observation history O_t, and $\gamma_{pla} \in [0, 1]$ is an exponential discount factor. The function $obs(x, k, O_t)$ simply returns 1 if state x can be observed by time $t + k$, and 0 otherwise,

$$obs(x, k, O_t) = 1 \text{ if } x \in V^+(O_t, k), \text{ or } 0 \text{ otherwise.} \tag{3.6}$$

At each time step, U_{pla} identifies which particles can feasibly be observed, and computes a weighted sum of their probabilities. Pursuers should select control actions maximize this value. The future location of each particle $x_{i,t+d}$ can be computed by forward simulating the particle system using some opponent model M. The discount factor γ_{pla} biases the heuristic towards the more immediate future.

When U_{pla} is used to select control actions, it will direct pursuers towards nearby locations that can observe the particles in X. We demonstrate empirically in Sec. 3.3 that this is an effective way to recover visibility on the evader when the contamination region is large. However, the computation of U_{pla} completely disregards the boundaries of the contamination region, so it considerably less effective at keeping the contamination region small in the first place.

In our implementation, pursuers will use U_{pla} only when the size of the contamination region exceeds some critical threshold $\tau_{recover}$. In all other cases, the pursuers fall back on U_{lel}, which complements the features of U_{pla}. In particular, U_{lel} is good at keeping the contamination region small, but ineffective once the region becomes sufficiently large. We show empirically how the selection of $\tau_{recover}$

affects the performance of the pursuer team in Sec. 3.3.

3.2.1 Complexity Analysis

The time complexity of updating particle system X is $O(t|X|)$ over t time steps. This is follows from the observation that updating a single particle's location is constant with regards to the size of the environment or any other variables. The re-sampling of particles can also be performed in time $O(|X|)$ when needed using the re-sampling algorithm described in [51].

The shortest path distance $d(y_i, y_j)$ from any state y_i to all states $y_j \in L_{grid}$ can be computed in time $O(|L_{grid}|)$ using the Fast-Marching Method (FMM) mentioned in Sec. 2.4. To compute potential field F_H as defined in Eqn. 3.3, the FMM must be performed once for each sample and pursuer, which takes time $O(m|L_{grid}|+n|L_{grid}|)$, where m is the number of samples and n is the number of pursuers. The complexity is reduced with respect to m if only the k-nearest samples are used. Evaluating only the k-nearest can be done in time $O(k|L_{grid}| + n|L_{grid}| + m \log k)$, where $O(m \log k)$ is the complexity of identifying the minimum k values in a set of size m.

Computing $eval(x, Y)$ for all $x \in L_{grid}$ can be done in time $O(n|L_{grid}|)$ using the caching method described in Sec. 2.4.3. Thus, the complexity using U_{pla} to evaluate a single control action is $O(t|X| + n|L_{grid}| + k|L_{grid}| + m \log k)$. In practice, the work performed by the first evaluation can be reused in subsequent evaluations, resulting in a lower amortized time.

3.3 Experimental Results

To evaluate the performance of the PLA heuristic and each of the opponent models, we performed a series of experiments using the randomly generated domains that were introduced in Chapter 2. We also compared our algorithm with the work of LaValle et al. [2, 7] in two case studies on hand-selected domains. The primary result of this analysis was that the PLA heuristic achieved better performance than the other approaches that were evaluated, independent of the choice of opponent model. Between the different models, informed models such as M_{LP}, M_{EV} and M_{EP} performed better than the uninformed models such as M_{RW} and M_{PRW}.

The experimental setup for these experiments closely resembles the Chapter 2 setup described in Sec. 2.5.1. For each experiment, 2000 randomly generated games were simulated between two pursuers and one evader. At the beginning of each trial, the evader was either placed at the boundary of the pursuers' visible area or, in the case of recovery time experiments, placed at a random location in L_{free} that was not visible to pursuers. Each trial was performed using one of the three evader strategies introduced in Sec. 2.5.2.

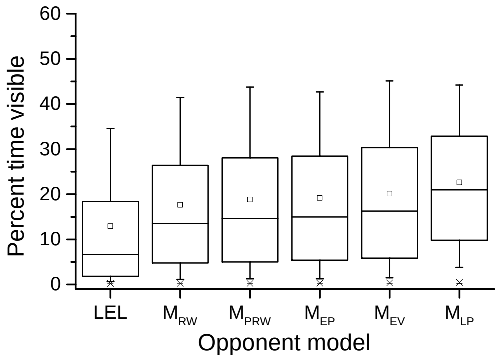
Except where otherwise specified, the parameters used by the PLA heuristic were as follows: particle change probability $p_{change} = 0.2$, particle change deviation $\sigma_{change} = 45°$, exponential decay constants λ_x and $\lambda_y = 0.05$, discount factor $\gamma_{pla} = 0.95$, recovery threshold $\tau_{recover} = \frac{|L_{free}|}{10}$, particle set size $|X| = 1000$, historical data set size $|H| = 2000$, and nearest neighbor sample size $k = 10$. These values were hand-selected based on what appeared to offer the best recovery time for the team of pursuers. Figures depicting the performance for different values of these parameters are shown in Sec. 3.3.3.

To gather the set of historical data H for the *learned preference* opponent model, 100 games with a duration of 200 time steps were performed prior to each of the experiments. This data collection was performed individually for each domain and evader strategy. We assumed that the pursuer was aware which evader it would be facing in advance. The pursuer strategy used while collecting historical data was PLA with the M_{RW} opponent model. H was sampled uniformly from the set of states where the evader was not visible. This was done to ensure that the sample data corresponded only to situations where the opponent model would be used.

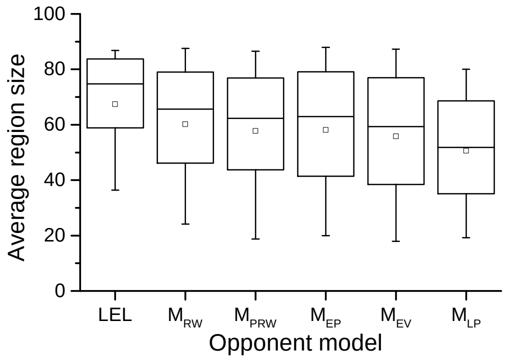
3.3.1 Performance Comparison

The first set of experiments were designed to compare the performance of pursuer teams using the LEL and PLA heuristics with different opponent models in randomly generated domains. The opponent models for the PLA heuristic introduced in this chapter include: *random walk* (M_{RW}), *persistent random walk* (M_{PRW}), *evade pursuer* (M_{EP}), *evade visibility* (M_{EV}), and *learned preference* (M_{LP}). Two of these models, M_{EP} and M_{EV}, correspond with the S_{EP} and S_{EV} evader strategies introduced in Sec. 2.5.2. The learned model, M_{LP}, was tuned for each evader using the technique described in Sec. 3.3.

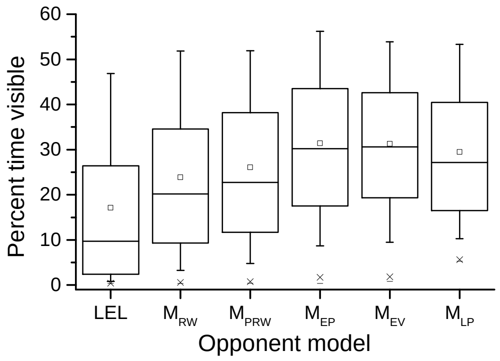
For these experiments, we simulated games where the initial location of the evader was known. As in Chapter 2, the evader was placed at the boundary of the



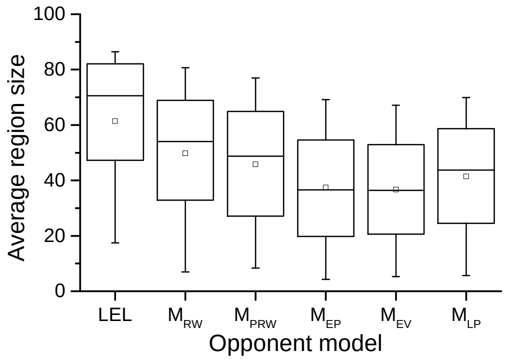
(a) Percent time visible vs. S_{RL} evader



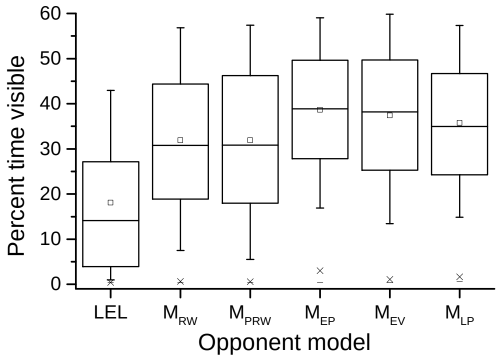
(b) Contamination region size vs. S_{RL} evader



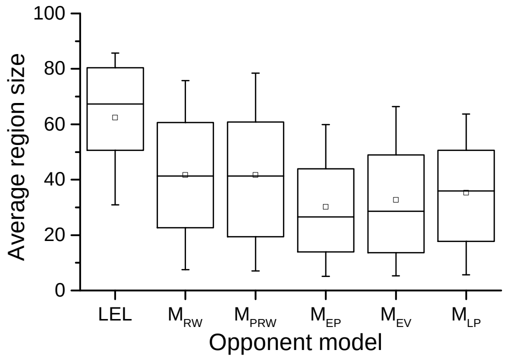
(c) Percent time visible vs. S_{EV} evader



(d) Contamination region size vs. S_{EV} evader



(e) Percent time visible vs. S_{EP} evader



(f) Contamination region size vs. S_{EP} evader

Figure 3.5: Comparison of LEL and PLA heuristics with different opponent models against the S_{RL}, S_{EV} and S_{EP} evader strategies: (left) percentage of time the evader was visible, higher is better (right) average size of the contamination region, lower is better. The box plots show the median, upper and lower quartile. The whiskers show the upper and lower decile. The mean value is represented by a square.

pursuers' visible area, and the speed of the evader was set to 10% faster than the pursuers. This ensured that the evader would eventually escape detection. Simulations were run for 1000 time steps and the average size of the contamination region and percentage of time the evader was visible were recorded.

The results in Fig. 3.5 show that the PLA heuristic outperformed the LEL heuristic against all three evader strategies. This result held true even when PLA was given a random model, such as M_{RW} and M_{PRW}, which required no prior knowledge about the evader's strategy. Between the different opponent models, the more informed models did better: M_{EP} and M_{EV} did best against the matching S_{EP} and S_{EV} evader strategies, while M_{LP} performed best against the S_{RL} evader strategy. Even against the S_{EP} and S_{EV} evaders strategies, M_{LP} came in third, not far from the performance of the matching models.

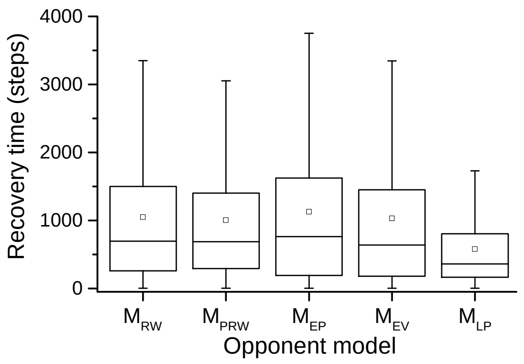
The learned model, M_{LP}, was only outperformed in cases where an exact model of the evader's strategy such as M_{EP} or M_{EV} was provided. The reason the M_{EP} and M_{EV} models both do well against S_{EP} and S_{EV} is that the behavior generated by those strategies is very similar; at far distances from the pursuers, the shortest path to the nearest pursuer and shortest path to the pursuers' visible region are approximately the same. Against the S_{RL} strategy, however, M_{EP} and M_{EV} were not much better than the random models.

One explanation for why the PLA heuristic performs better than the LEL heuristic, even using the random opponent models, is that the PLA heuristic at a minimum induces the pursuers to explore the environment. Using the random models, the likelihood that an evader is located in various parts of the environment is approximated by the “window of opportunity” available for particles to travel there. If an area has recently been explored by the pursuers, it will not be populated by as many particles, and the pursuers will explore other parts of the environment until that area repopulates. The LEL heuristic on the other hand is comparatively over-conservative, treating all locations as equally probable, leading to the “stalling” behavior discussed in Sec. 2.5.3.

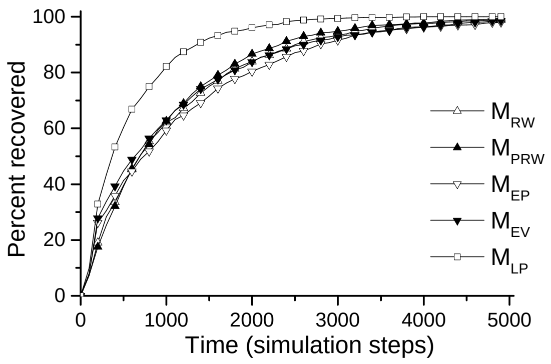
3.3.2 Recovery Time Comparison

The second set of experiments were designed to evaluate the recovery time associated with each opponent model. For these experiments, we simulated games where the initial location of the evader was not known. In contrast to the previous experiments, the evader was initially placed at a random location outside the pursuers’ visible area. The contamination region, representing the pursuers’ belief about the evader, was initialized to all locations not initially visible to the pursuers. The particle set X was initialized to a random set of locations sampled uniformly from this region. Each simulation was run for 5000 time steps, and the time at which the pursuer team regained visibility of the evader was recorded.

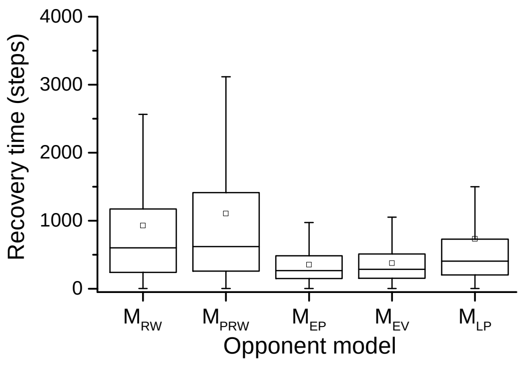
The results in Fig. 3.6 show that PLA using the informed opponent models performed better than PLA using the random opponent models. The M_{EP} and M_{EV} models had the shortest recovery times against the S_{EP} and S_{EV} evader strategies, while the M_{LP} model had the shortest recovery time against the S_{RL} evader strategy. Interestingly, the M_{EP} and M_{EV} models did worse than the random models against



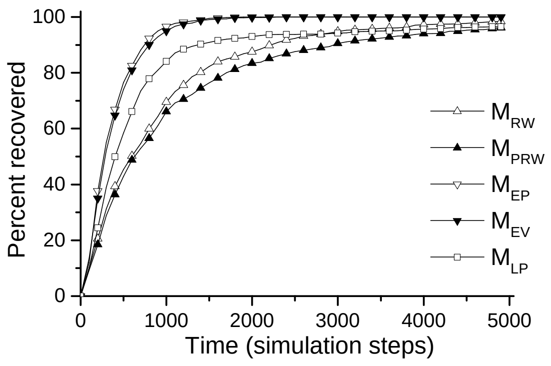
(a) Percent time visible vs. S_{RL} evader



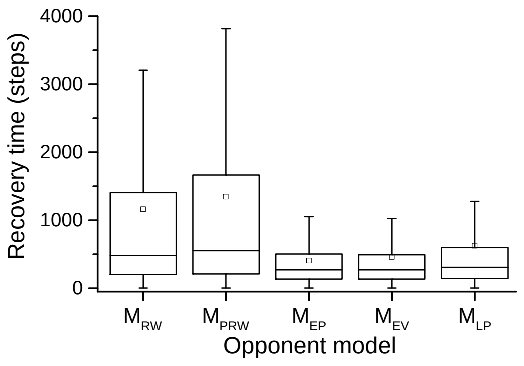
(b) Contamination region size vs. S_{RL} evader



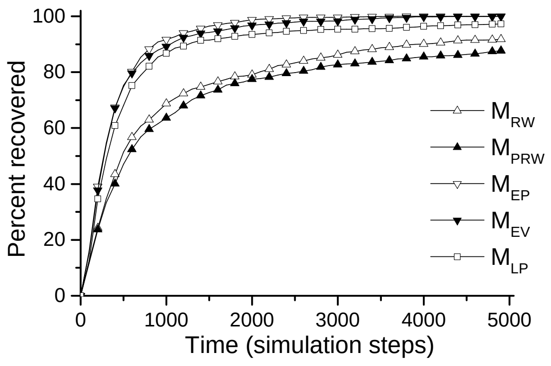
(c) Percent time visible vs. S_{EV} evader



(d) Contamination region size vs. S_{EV} evader



(e) Percent time visible vs. S_{EP} evader



(f) Contamination region size vs. S_{EP} evader

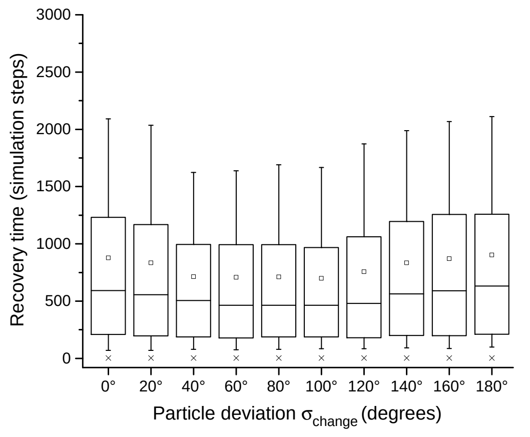
Figure 3.6: Comparison of PLA heuristic with different opponent models against the S_{RL}, S_{EV} and S_{EP} evader strategies: (left) average amount of time it takes the pursuers to recover an evader that is initially hidden, lower is better, (right) the same data represented as a cumulative measurement over time.

the S_{RL} evader strategy, suggesting there is a penalty associated with using the incorrect model. An explanation for the differences in performance between each of these models follows the same justification provided in Sec. 3.3.1. The informed models do better because they more effectively estimate the probability distribution over where the evader is located.

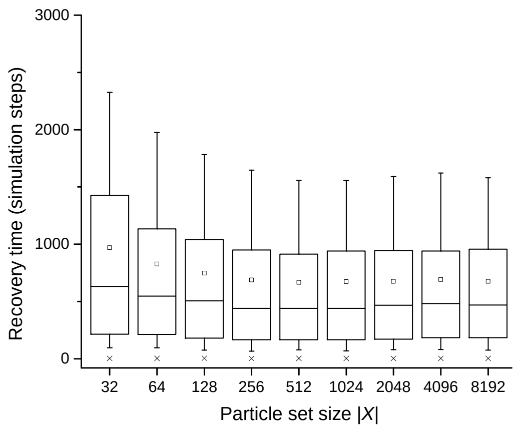
3.3.3 Parameters for Learned Model

The following set of experiments were designed to compare the performance of the PLA heuristic using the M_{LP} opponent model for a variety of different parameter values. The strategy used by the evader in these experiments was S_{RL}, the most challenging evader strategy from Sec. 3.3.1. The parameters that were varied include particle deviation σ_{change}, particle set size $|X|$, historical data set size $|H|$, nearest neighbor sample size k, and recovery threshold $\tau_{recover}$. For each experiment, except the recovery threshold experiment, we simulated games where the initial location of the evader was not known. Each simulation was run for 5000 time steps, and the time at which the pursuer team recovered visibility of the evader was recorded.

The results in Fig. 3.7a show that the fastest recovery times are achieved when the value σ_{change} lies between 40 and 100 degrees. Recall that σ_{change} is the standard deviation of the Gaussian from which ψ_X is sampled, where ψ_X is the random angle by which particles' control actions are rotated. There is an intuitive explanation for why this "sweet spot" exists. Very high values of σ_{change} negate the benefit of using the learned potential field at all; particles will move entirely at random, so



(a)



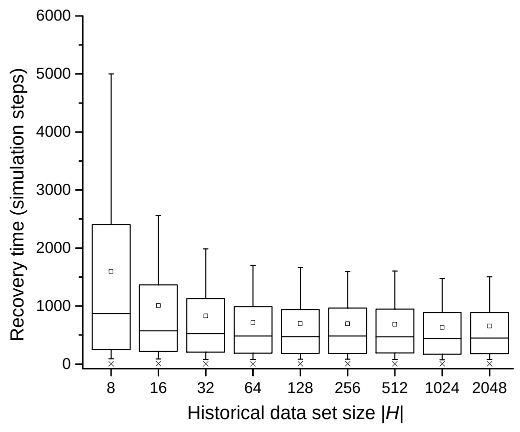
(b)

Figure 3.7: (a) Average recovery time for team of two pursuers as the particle deviation σ_{change} is varied. (b) Average recovery time for team of two pursuers as the particle set size $|X|$ is varied. Lower values indicate better performance.

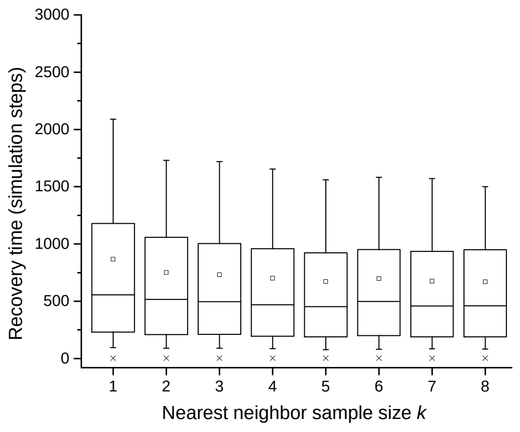
it is equivalent to just using M_{RW}. On the other hand, very low values of σ_{change} create a less diverse set of hypotheses about the evader, poorly generalizing from the training set and resulting in overfitting.

The results in Figs. 3.7b, 3.8a, and 3.8b show that increasing the sizes of the particle set X, historical data set H or nearest neighbor sample size k are all effective in lowering the recovery time. However, in all three cases, there is a point of diminishing returns after which the additional benefit becomes insignificant. The most modest results are obtained by increasing the nearest neighbor sample size k, where benefits become insignificant after $k = 5$.

For the recovery threshold experiment, we simulated games where the initial location of the evader was known, as was done in Sec. 3.3.1. Each simulation was run for 1000 time steps, and the percentage of time the evader fell within the pursuers' visible area was recorded. Recall that the recovery threshold value $\tau_{recover}$ determines how large the contamination region can become before the PLA heuristic will start



(a)



(b)

Figure 3.8: (a) Average recovery time for team of two pursuers as the historical data set size $|H|$ is varied. (b) Average recovery time for team of two pursuers as the nearest neighbor sample size k is varied. Lower values indicate better performance.

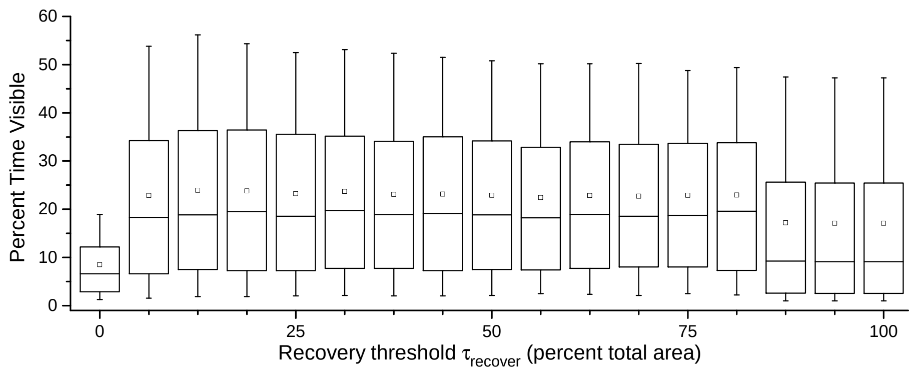


Figure 3.9: Average amount of time the evader was visible in randomly generated games with two pursuers as the recovery threshold $\tau_{recover}$ is varied. Higher values indicate better performance.

to use U_{pla} instead of U_{lel} for control action selection.

The results in Fig. 3.9 show that the selection of $\tau_{recover}$ can have a significant effect on the performance of the pursuers, but only at extreme values. The best results are obtained when $\tau_{recover}$ is roughly between 10% and 75% of the total area of the environment. The heuristic starts to do poorly when $\tau_{recover}$ is lower than 10% of total area, a consequence of U_{pla} being used almost exclusively in this case, even though U_{lel} is better than U_{pla} at keeping the contamination region small. When

$\tau_{recover}$ is greater than 75% of the total area, U_{pla} is rarely used at all, and the performance becomes equivalent to just using LEL. There is very little difference between 10% and 75% because the contamination region is usually either very large or very small, so any value within that range has roughly the same effect.

3.3.4 Case Studies

In this section, we evaluate the performance of the PLA heuristic for two problem domains selected from the work of LaValle et al. [2, 7]. We compare the recovery time performance of PLA with the performance of the algorithm introduced by LaValle et al. in that same work. For the purposes of convenience, we will refer to this algorithm as the LV algorithm.

Recall from Sec. 2.1 that the LV algorithm was designed exclusively for domains where the initial location of the evader is unknown. The LV algorithm works by decomposing the environment into convex subregions and performing an exponential-time search over the problem's information-space. The LV algorithm is complete in the sense that it is guaranteed to find a path that detects the evader, assuming such a path exists when the speed of the evader is infinite.

The LV algorithm's success or failure is measured by its ability to completely clear the contamination region. For this reason, LaValle et al. did not specify an actual evader strategy to test against. In these experiments, we use the S_{RL} evader strategy specified in Sec. 2.5.2. Similar results are obtained using either of the other two strategies, but the variance between different approaches is smaller due to the

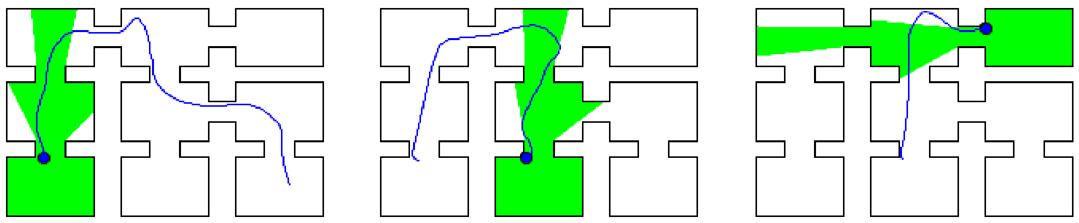


Figure 3.10: Example path for a single pursuer using the PLA heuristic with M_{EV} opponent model in example domain 1. The shaded region represents the area visible to the pursuer. Images are ordered chronologically from left to right.

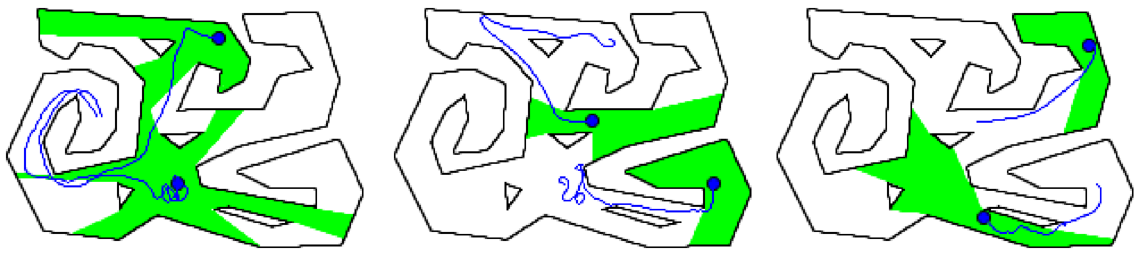
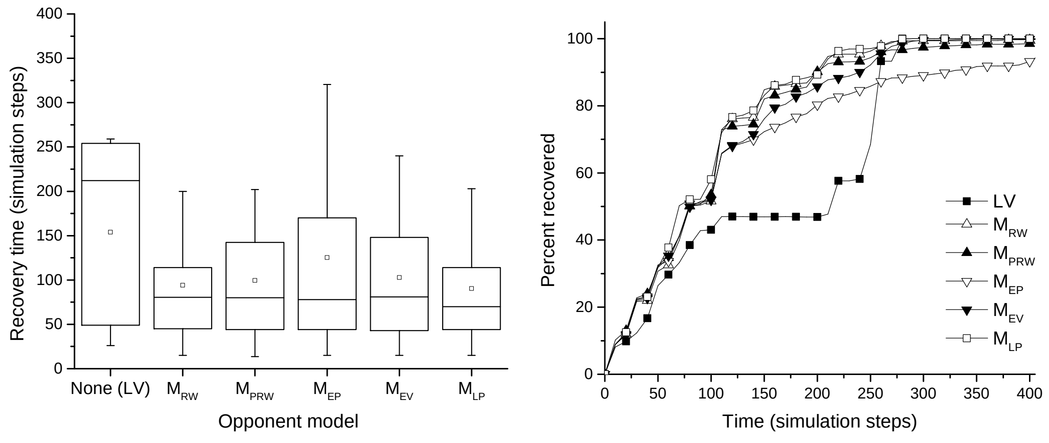


Figure 3.11: Example paths for two pursuers using the PLA heuristic with M_{EV} opponent model in example domain 2. The shaded region represents the area visible to the pursuer. Images are ordered chronologically from left to right.

small size of the domains.

The LV algorithm places restrictions on the type of domains that can be solved. In particular, any domain that has a topology with h holes requires $\Omega(\log h)$ or more pursuers to clear the contamination region [7]. The LV algorithm also requires that the visibility range of the pursuers be infinite, which differs from our experiments in Secs. 3.3.1 and 3.3.3. We have satisfied both of these restrictions for our experiments in this section. The setup for these experiments is the same as the recovery time experiments in Sec. 3.3.2, except the visibility range is infinite, and we use two domains for which the LV algorithm is known to have a solution.

Example domain 1, shown in Fig. 3.10, consists of a set of nine interconnected rooms which can be successfully cleared by a single pursuer using the LV algorithm.



(a) (b)

Figure 3.12: (a) Average recovery time for a single pursuer using the LV and PLA strategies to search for an S_{RL} evader in example domain 1. (b) The same data represented as a cumulative measurement over time.

The results in Fig. 3.12 show that the fastest strategy in this domain was PLA using the M_{LP} opponent model, with a mean recovery time of 93 time steps. The slowest was the LV algorithm with a mean recovery time of 153 time steps. There was very little difference between the M_{LP} and M_{RW} models in this scenario, possibly attributable to the small size of the domain. Of the other models, the worst performing was M_{EP}, consistent with the results in Sec. 3.3.2 indicating that M_{EP} is a poor model of the S_{RL} strategy.

Example domain 2, shown in Fig. 3.11, consists of a set of irregular pathways with random obstacles positioned throughout. This domain requires two pursuers to successfully clear using the LV algorithm. The results in Fig. 3.13 show that the fastest strategy was again PLA using the M_{LP} opponent model, with a mean recovery time of 207 time steps. The slowest was the LV algorithm with a mean recovery time of 345 time steps.

In both domains, the difference in performance between the LV algorithm

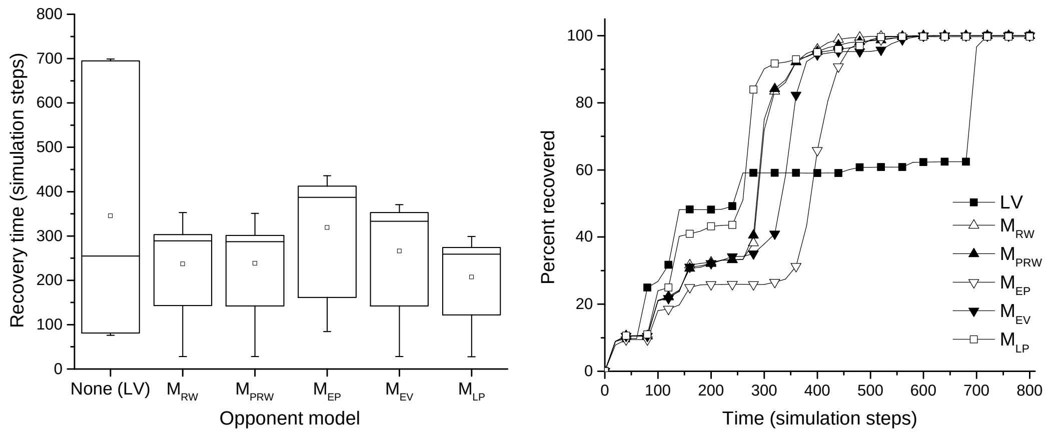


Figure 3.13: (a) Average recovery time for a team of two pursuers using the LV and PLA strategies to search for an S_{RL} evader in example domain 2. (b) The same data represented as a cumulative measurement over time.

and the PLA heuristic can be attributed to the comparatively conservative nature of the paths generated by the LV algorithm. The paths are carefully chosen to avoid recontamination and do not exploit the probabilistic model of the evader. In contrast, the PLA heuristic relies exclusively on the probabilistic model and is not concerned with recontamination of the environment.

Although PLA achieved the fastest recovery time in these two domains, this result is specific to the opponent strategy used. Given a theoretically worst-case outcome, such as the one assumed by LaValle et al., there is no guarantee that PLA would ever recover visibility on the evader. The only way to provide that guarantee is by completely clearing the contamination region, something the LV algorithm is specifically designed to do. For domains where this goal is explicitly desired, the LV algorithm is a better choice than the LEL heuristic.

The primary advantage of PLA is its applicability to domains where the LV algorithm cannot provide a solution. In particular, LV is not applicable to the vast majority of the randomly generated domains used in Secs. 3.3.1 and 3.3.3, since the topology of those domains prohibits the LV algorithm from finding paths that can completely eliminate the contamination region. In contrast, the PLA heuristic is applicable to those domains because it assumes that the opponent's behavior can be modeled probabilistically. In addition, PLA places very few restrictions on the sensor model used by the pursuers, while LV requires that the sensor range be infinite. This flexibility, combined with the comparatively low computational complexity of PLA, suggest that the PLA heuristic may be a better choice than the LV algorithm for a number of scenarios.

The recovery time of the LV algorithm could be improved by incorporating probabilistic information about the evader, an extension that was explored by Stiffler and O'Kane [6]. However, the model in that work relied solely on prior probabilities, and the authors did not specify how to obtain the model. In future research, it might be worth investigating how to combine the M_{LP} opponent model with the LV algorithm and see how the recovery times compare with PLA. However, the LV algorithm would still be limited in the domains it can be applied.

3.4 Discussion

This chapter introduced the *probabilistic lookahead* (PLA) heuristic, a control action selection heuristic for the pursuer team that evaluates control actions by forward simulating the motion of the evader using a probabilistic opponent model. Four opponent models were introduced, including two random opponent models, and a *learned preference* (M_{LP}) opponent model trained using historical data about the evader's location. Experimental results showed that using these opponent models PLA is effective at recovering visibility of evader in situations of high uncertainty, addressing a significant weakness of the LEL heuristic from Chapter 2. Results also demonstrated that PLA is able to recover the evader more quickly than the algorithm by LaValle et al. [7] in two case studies on example domains.

Across all of the experiments, the best performance was obtained when the PLA heuristic was used in conjunction with an informed opponent model. The random walk opponent model provided adequate performance, outperforming LEL and the algorithm by LaValle et al. in the example domains. In situations where historical data about the opponent's location is not available, the random models may serve as an effective substitute for the learned preference model.

3.4.1 Generalizations of PLA

The use of particle filters for modeling purposes is not new [45, 52], however, the primary novelty of PLA is that opponent models can be encoded as potential fields generated from historical observations. The relative simplicity with which this model can be generated leads to a variety of potential applications in both cooperative and adversarial situations.

Although the configuration space in this chapter was restricted to $\mathbb{R}^2$, there is no reason why this approach is limited to two-dimensional domains. The potential fields generated in Sec 3.1.4 are implemented on top of the Fast Marching Method which, as discussed earlier, can be extended into higher dimensions. Particles can also be extended to higher dimensions, although more particles are required as the dimensionality increases [45].

The primary limitation of potential fields when modeling an agent's movement are their expressibility. With the potential field model, particles are attracted to local minima only, so a single potential field cannot easily represent opposing hypothesis about where a local set of particles will travel. A possible extension of this approach would be to split the model across multiple potential fields when dramatically opposing hypotheses are identified in the learning phase.

3.4.2 Deploying in a Real-World Setting

The efficient running time and reasonable performance of the LEL and PLA heuristics are encouraging signs that this approach may be feasible for real-world application. However, a number of additional challenges must be overcome before deploying on a robotic platform is realistically feasible. Several of the most important challenges are discussed below.

Imperfect Localization: While a variety of robot localization techniques exist in literature, most noticeably FastSLAM and its variants [47,48], the output of these algorithms is typically a probabilistic weighting over candidate locations. The most straight-forward way to incorporate probabilistic localization into the PLA heuristic is to evaluate each candidate location separately, then select a control action which maximizes weighted average of utility across all candidates. However, this requires k-times the computational effort in scenarios with k-candidates. The PLA heuristic can be computed quickly enough that this technique may be feasible, but the tradeoff between number of candidates evaluated and performance should be evaluated experimentally in future work.

Non-Holonomic Motion Constraints: Many robotic platforms are subject to differential motion constraints, such as limited acceleration or turning radius. The Fast Marching Method is designed primarily with the assumption that motion is holonomic, so an alternative method must be chosen. One possible approach is to predict the set of future states for the agents using particle filters, as is done for the target boat in Ch. 4. Another alternative are Fast Marching Trees, a relatively new technique for modeling state propagation over more complex configuration spaces [43]. An experimental evaluation the tradeoff of different techniques should be performed as part of future research.

Noisy Sensor Data: In both Ch. 2 and Ch. 3 the sensor region of the pursuers is presented as a solid polygonal boundary. In the real-world, sensor readings are noisy, and the ability of pursuers to detect an evader within a particular region should be characterized probabilistically. Fortunately, particle systems provide us with a straight-forward method of modeling this situation: rather than completely discarding a hypothesis when it intersects the sensor region, the probability can simply be discounted. How much the hypothesis should be discounted will depend on the capabilities of the sensor, which needs to be modeled before being incorporated into the algorithm.

Incomplete Map: One major assumption of the work in this dissertation is that an accurate map of the obstacles is provided in advance. How the algorithm should model locations and obstacles that have not yet been observed is perhaps the largest open question regarding how the algorithm can be applied in the real-world. Even for obstacles that have been observed, the accuracy of the map is subject to the sensor limitations of the robot. One approach may be to model the unobserved portions of the environment as if they have perfect connectivity: i.e., if no obstacle has been observed, assume that none exists. It is unlikely that this approach will yield optimal performance, but it is a reasonable starting point until more sophisticated prediction methods are developed. How the algorithm performs when there are inaccuracies in the map should be evaluated empirically as part of future work.

3.4.3 Limitations and Future Work

One of the assumptions of the learned preference opponent model is that the behavior of the evader is conditioned on some state vector Y representing the current location of the pursuers. This assumption was made because the strategy used by the actual opponent in our experiments was conditioned on that same information. In practice, the evader may not have access to the current location of the pursuers, and a different input vector may be appropriate. For example, the evader strategy may be conditioned on the most recently observed locations of the pursuers, or some other attribute such as time. It should be possible to generalize the model to handle arbitrary input vectors, which is a subject for future research.

Another assumption of the learned preference opponent model is that historical data is specific to individual environments. The model does nothing to generalize observations from past environments to new environments. This assumption must be relaxed for the approach to be used in environments that have not previously been encountered. One possible approach is to classify historical observations of the evader in terms of abstract features of the environment rather than exact locations. Due to the success of purely random opponent models, this approach may work even if the information gleaned about the evader is minimal. However, the exact method for doing this and its efficacy are a subject for future research.

Chapter 4: Target Following with Unmanned Boats

This chapter introduces a motion goal selection algorithm for pursuit-evasion games between two marine vessels subject to differential motion constraints. The pursuer vessel, which may be an unmanned sea surface vehicle (USV), must closely follow the evader while safely navigating an obstacle field. Example applications of this problem include naval escorts through a crowded harbor, wildlife monitoring in marine protected areas, or pursuing criminals through shallow coastal regions.

The motion goal selection heuristic introduced in this chapter utilizes an independent, *black box* path planning algorithm to identify dynamically feasible paths and calculate their travel cost. The heuristic performs Monte-Carlo sampling over a probabilistic opponent model to select candidate motion goals, which are then evaluated using a cost function that incorporates the travel costs computed by the path planner. Experimental results show that motion goals selected using this technique reduce the pursuer's travel distance and increase the percentage of time the pursuer is within range of the evader when compared to simpler approaches that use the current location of the evader as a motion goal.

While the previous two chapters demonstrated how problem relaxation and Monte Carlo sampling can be used to select control actions in pursuit-evasion games, there are circumstances where it may be preferable for a planner to generate high-level motion goals independently from low-level control actions. For example, the control system for a USV may calculate control actions using a control-loop feedback mechanism, such as a PID controller [53]. Additionally, separating motion goal selection from control action selection allows for intermediary planning phases that handle local obstacle avoidance or process dynamically feasible paths [54,55]. This is particularly useful when calculating a dynamically feasible path requires simulating complicated motion dynamics. It is with that objective in mind that we develop the algorithm introduced in this chapter.

4.1 Background

This chapter builds upon the work of Švec et al. which introduces a trajectory planning algorithm able to compute dynamically feasible paths for USVs in marine environments with obstacles [53, 55, 56]. The planner uses a lattice-based discretization of the USVs' configuration space to perform a A* search over the space of feasible trajectories. The planner also independently computes the desired velocity of the USV to satisfy the specified arrival time for a motion goal. This algorithm is utilized by our motion goal selection heuristic to determine the travel cost to reach individual candidate motion goals.

A survey of existing state-of-the-art approaches for target following is provided by Bibuli et al. [57]. This includes multi-vehicle motion control. In addition, the paper presents experimental validation of USV following a leader vessel by observing its path and precisely executing it using a path-following algorithm.

High-speed straight-line tracking capability was developed in [58] which allows underactuated USVs to follow a moving target. Based on the guidance system previously utilized for interceptor missiles, the motion control system is composed of constant bearing guidance and velocity control schema that allows high and precise maneuverability.

A variety of advanced maneuvers for searching and tracking a target, docking, reactive collision avoidance, U-turn, course tracking, and waypoint following are implemented on the MESSIN system [59]. The system is capable of handling failures of its components through various emergency programs. In addition, the integrated path planning utilizes waypoints and motion primitives represented as circular arcs.

In order to handle motion uncertainties due to ocean disturbances, a dynamic surface control and adaptive formation controller represented as a neural network was developed in [60]. Similarly, a formation control algorithm for following a master vessel while considering uncertain dynamics of the vessel can be found in [61].

4.2 Problem Formulation

In this chapter, we specify a pursuit-evasion scenario between a single unmanned sea-surface vehicle (USV) and a target boat which may or may not be evasive. Unlike the visibility-based pursuit-evasion formalism introduced previously, we are not concerned with maintaining visibility on the target. Rather, we assume that the USV is continuously aware of the target boat's location, and that its objective is to position itself behind the target while safely avoiding obstacles.

The formal definition of the problem includes:

(i.) a vehicle state space $X \subseteq \mathbb{R}^3 \times \mathbb{S}$, where each state $x = \langle l, \psi, v \rangle$ defines the location l, heading ψ, and surge speed v of a single boat or USV,

(ii.) a control action space $Q(x) \in \mathbb{R}^2$ for each $x \in X$, where each control action $q = \langle \Delta v, \Delta \psi \rangle$ defines a change in surge speed Δv and heading $\Delta \psi$,

(iii.) an opponent model, $M_B(x_{t+k}|x_t, O_t)$ defining the probability that the target boat will transition from state x_t to x_{t+k} given observation history O_t,

(iv.) an obstacle field Ω, where collision with any obstacle $\omega \in \Omega$ results in an immediate failure for the USV

(v.) a minimum and maximum desired radius r_{min} and r_{max} defining a circular proximity region X_P around the target

The performance of the USV is measured in terms of how consistently it stays within the desired range of the target, as well as its total energy expenditure in terms of distance traveled. The ideal pursuit strategy should maximize the time that the USV is within proximity region X_P and minimize the travel distance.

For the motion of boats and USVs, we use a simple steering model where control action $q = \langle \Delta v, \Delta \psi \rangle$ determines the change in surge speed v and heading ψ, while $\Delta l_x = v \cos(\psi)$ and $\Delta l_y = v \sin(\psi)$ determine the change in the coordinates of location $l = \langle l_x, l_y \rangle$. To estimate the movement of the USV after time step Δt, we use the formula

$$x' = x + \Delta x(q)\Delta t \tag{4.1}$$

where $\Delta x(q) = \langle \Delta l, \Delta\psi, \Delta v \rangle$ represents the change in vehicle state x given control action q. The set of control actions $Q(x) = A(x) \times \Theta(x)$ is subject to physical motion constraints, where

$$A(x) = \{\Delta v : a_{min}(x) \leq \Delta v \leq a_{max}(x)\} \tag{4.2}$$

$$\Theta(x) = \{\Delta\psi : \theta_{min}(x) \leq \Delta\psi \leq \theta_{max}(x)\} \tag{4.3}$$

define the minimum and maximum change in surge speed Δv and turning angle $\Delta\psi$ given vehicle state x. We assume that the maximum turning radius decreases as the surge speed v increases. Each boat has a maximum surge speed, v_{max}, which it cannot accelerate past. We also assume that boats cannot travel in reverse, so $a_{min}(x)$ is zero when the surge speed v is zero.

We assume that the USV has access to an evaluation function $eval(x)$ which returns the travel time for the USV to reach state x from its current state x_u. This function is computed using the lattice-based A* pathfinding algorithm originally described by Švec et al. [55]. If state x is unreachable, or would place the USV in a region of inevitable collision, then $eval(x)$ will return infinite cost. The motion goal selection algorithm described in the following section utilizes this function in its evaluation of candidate motion goals.

4.3 Motion Goal Prediction

To maintain a suitable distance between the USV and the target boat, we introduce heuristic motion goal prediction algorithm designed to calculate the desired motion goal x_g and arrival time t_g for the USV. The algorithm works by estimating the future poses of the target boat based on probabilistic opponent model M_B and then evaluating several candidate motion goals for the USV. Since the motion goal must be computed relatively quickly, we search for a sub-optimal solution by combining Monte-Carlo sampling and heuristic evaluation techniques. The process is described below and summarized in Alg. 3.

Opponent model M_B defines state transition probabilities over the set of target poses $x_i \in X$. To explore the future states of the target, we sample n random paths $f_j \in F$ starting from the target's current pose and forward-projecting the target's actions up to some finite time horizon t_d. During the sampling process, each state transition is selected with probability $M_B(x_{t+k}|x_t, O_t)$. Sampled paths that lead to a collision with an obstacle are discarded from F.

By recording the poses reached by the target boat along each sampled path, we can estimate the probability $P_{b,i}[l_j]$ that the target will be at position l_j at time t_i by counting the frequency of occurrence. To do this, we first discretize the problem space into a set of locations L_{grid} as was done in Chapter 2. We then apply a two-dimensional Gaussian kernel to smooth out the probability distribution represented by $P_{b,i}$, as illustrated in Fig. 4.1a. Similarly, we compute $\psi_{b,i}(l_j)$, which is the average heading at l_j across all samples at time t_i.

Algorithm 3 COMPUTEMOTIONGOAL()

```
Require: The current poses $x_b$ and $x_u$ of the target boat and USV, a probabilistic
    model of the target boat $M_B$, and a map of obstacles
Ensure: A desired motion goal $x_g$ and arrival time $t_g$.
 1: Let $F$ be a set of $n$ randomly generated paths for the target boat, where each
    path $f_i \in F$ begins at state $x_b$ and time $t_0$ and continues until time $t_d$, such that
    each state transition is sampled with probability $M_B(x_{t+k}|x_t, O_t)$.
 2: for each time point $t_i \in \{t_1, t_2, \ldots t_d\}$ do
 3:     for each path $f_j \in F$ do
 4:         Increment $P_{b,i}[f_j(t_i)]$ by $1/n$, where $f_j(t_i)$ is the location of path $f_j$ at $t_i$.
 5:     end for
 6:     Smooth $P_{b,i}$ by applying an $m \times m$ Gaussian kernel.
 7:     Let $l_i^*$ be the location that maximizes $P_{b,i}[l_i^*]$
 8:     Let $\psi_i^*$ be the mean heading at $l_i^*$ across all $f_j \in F$ at time $t_i$.
 9:     Let $x_{g,i} = \langle l_j, \psi_j \rangle \in X_g$ be a candidate motion goal for the USV at $t_i$ minimizes
        the distance to the projected state, $||l_i^* - l_j|| + \alpha||\psi_i^* - \psi_j||$
10: end for
11: Let $x_g^*$ equal the candidate motion goal $x_{g,i} \in X_g$ that minimizes the cost func-
    tion $cost(x_g)$
12: Compute a desired arrival time $t_g(x_g^*)$ such that the USV will arrive at $x_g^*$ shortly
    after the target boat
13: return  Motion goal $x_g^*$ and arrival time $t_g(x_g^*)$.
```

For each time point $t_i \in \{t_0, t_1, \ldots t_d\}$, the most likely location of the target boat at time t_i can be estimated by the formula

$$l_i^* = \arg\max_{l_j} P_{b,i}[l_j], \tag{4.4}$$

where l_i^* is the location with the highest probability density in $P_{b,i}$ after applying the Gaussian kernel. Similarly, the most likely heading of the target boat can be estimated by $\psi_i^* = \psi_{b,i}(l_i^*)$. We select the candidate motion goal $x_{g,i} = \langle l_i, \psi_i \rangle$ that is nearest the projected target pose $\langle l_i^*, \psi_i^* \rangle$ at time t_i, such that

$$x_{g,i} = \arg\min_{x_{g,j} \in X} ||l_i^* - l_j|| + \alpha||\psi_i^* - \psi_j|| \tag{4.5}$$

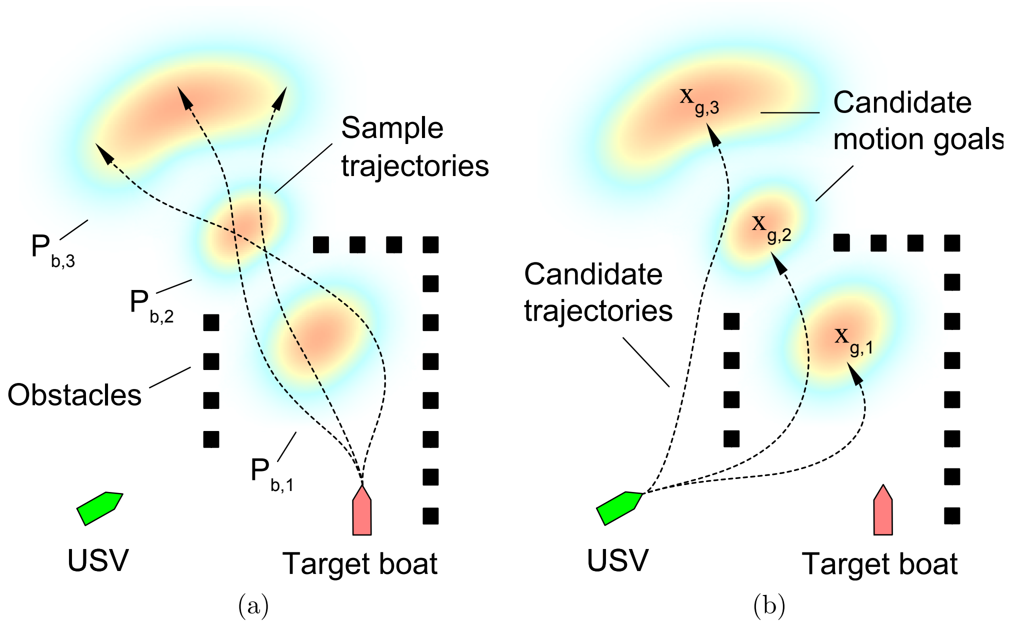


Figure 4.1: Motion goal prediction steps (a) Probability distributions $P_{b,1}$, $P_{b,2}$ and $P_{b,3}$, generated by sampling future target poses at each time point. (b) Candidate motion goals $x_{g,1}$, $x_{g,2}$ and $x_{g,3}$, generated by selecting the most probable target pose at each time point.

where α is a coefficient determining how heavily to weight differences in heading, and $||\psi_i^* - \psi_j||$ is the normalized interior angle between ψ_i^* and ψ_j.

We now have a set of candidate goals X_g, where each $x_{g,i} \in X_g$ represents the most likely pose of the target boat at time t_i. Let $eval(x_{g,i})$ be the amount of time it takes the USV to travel from its current state x_u to candidate goal $x_{g,i}$. This can be computed by the A* search process described in Sec. 4.2.

We define the cost function for motion goal $x_{g,i}$ as,

$$cost(x_{g,i}) = \begin{cases} \gamma^i(eval(x_{g,i}) - t_i + 1), \text{ if } eval(x_{g,i}) > t_i \\ \gamma^i, \text{ otherwise,} \end{cases} \tag{4.6}$$

where $eval(x_{g,i}) - t_i$ is the estimated difference in arrival time between the USV and target boat, and $\gamma \in [0, 1]$ is an exponential discount factor. The smaller that γ is, the stronger the bias towards earlier goals. From the set of candidate motion goals X_g, a final motion goal x_g^* is selected such that the cost function is minimized,

$$x_g^* = \arg \min_{x_{g,i} \in X_g} cost(x_{g,i}). \tag{4.7}$$

Given final motion goal x_g^*, the desired arrival time t_g is determined by,

$$t_g(x_g^*) = \begin{cases} eval(x_g^*), & \text{if } eval(x_g^*) > t_k + t_{lag} \\ t_k + t_{lag}, & \text{otherwise,} \end{cases} \tag{4.8}$$

where t_{lag} is the desired amount of time that the USV should follow behind the target boat. This ensures that the USV will reduce its speed when it can afford to do so, such as when it is already close the target boat.

Due to the design of the evaluation function $eval(x_g)$, the algorithm only considers motion goals which guarantee a collision-free path for the USV. States which lead to an inevitable collision are given infinite cost, as described in Sec. 4.2. If no collision-free paths are found, the USV will continue to use its currently assigned motion goal instead. For simplicity, we may select motion goals from the sample poses in F if the dynamics of the USV and target boat are equivalent, since the paths in F are already guaranteed to be collision free.

4.4 Experimental Results

To evaluate the motion goal selection algorithm described in Sec. 4.3 we conducted a series of experiments using randomly generated test cases. Each test case consisted of a set of 48 to 144 randomly placed obstacles in an environment with an area of 20×20 meters. We generated 200 different environments with five different obstacle densities, for a total of 1000 different test cases. In each test case, the target boat followed a randomly generated path sampled from the probabilistic model M_B. Example paths are shown in Figs. 4.2 and 4.3.

For comparison purposes, we also evaluated the performance of USVs which simply chose the current state of the target boat as a motion goal. In our results, we refer to this as the *simple* strategy, and refer to the motion goal selection algorithm introduced in Sec. 4.3 as the *heuristic* strategy. For each simulated trial, we recorded the amount of time the USV was within range of the target boat, and the length of the path followed by the USV.

4.4.1 Experimental Setup

At the start of each test case, the USV is positioned within 10 m of the target boat, and oriented at a heading facing the target. During the simulation, the target boat follows its pre-defined path for 2 minutes, while the USV follows a path generated by applying the control policy from Švec et al. [55] to the chosen motion goal. The path planner used to calculate $eval(x)$ was configured to terminate after 5000 iterations to reduce its running time. If no feasible path was found to any of

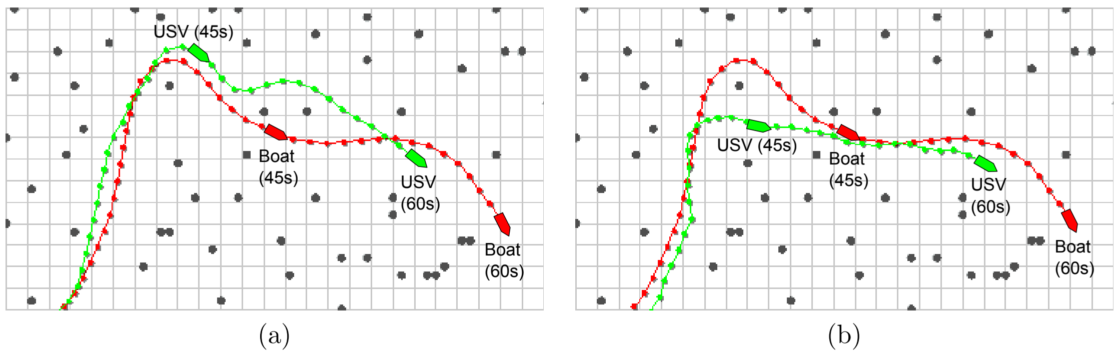


Figure 4.2: (a) Path of a USV using the location of the target boat as a motion goal. (b) Path of the USV using the heuristic motion goal selection algorithm introduced in Sec. 4.3. The path generated by the heuristic strategy is shorter.

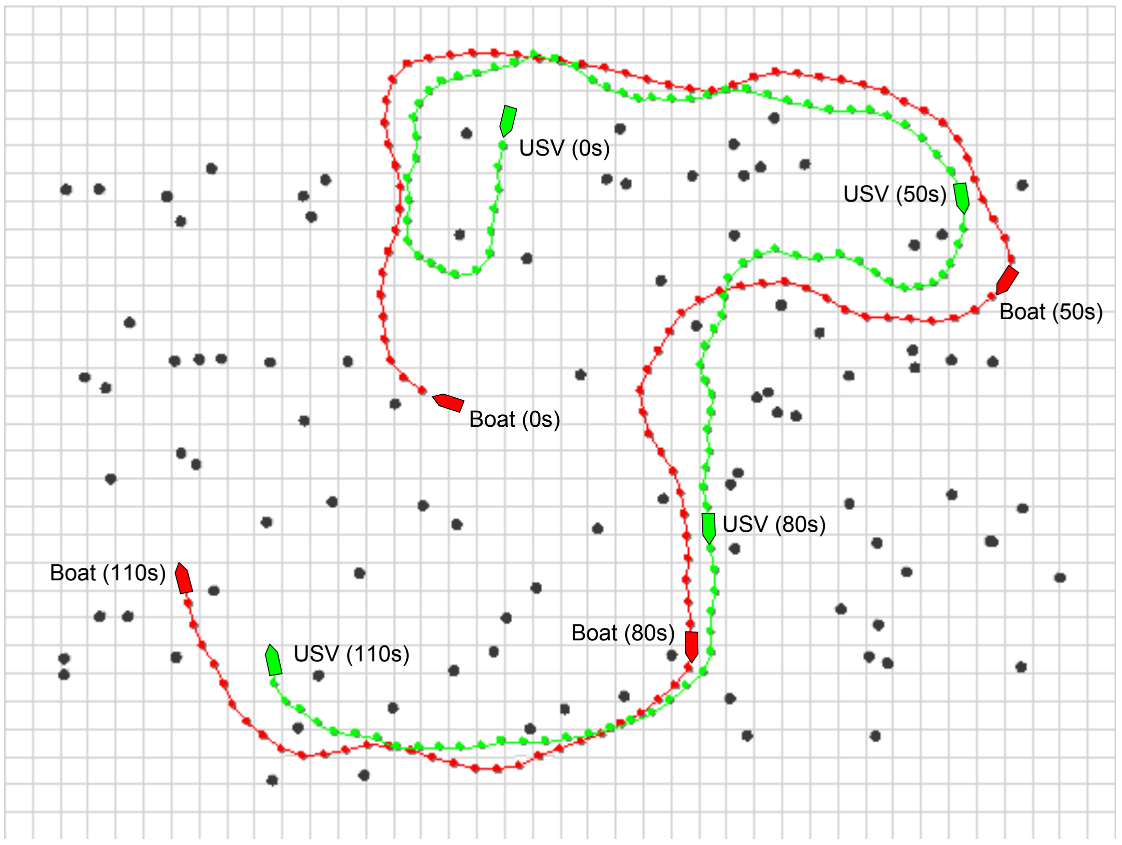


Figure 4.3: Paths of the USV and target boat in a randomly generated scenario with 96 obstacles. The path of the USV is generated by applying the control policy from Švec et al. [55] to motion goals selected by the algorithm introduced in Sec. 4.3. The path of the target boat is sampled randomly from opponent model M_B.

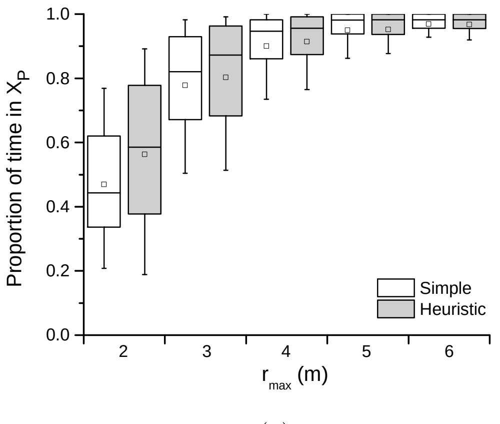
the candidate motion goals, then the previously computed path was used.

During each of the experiments, the parameters of the algorithm were as follows: the time horizon of the forward simulation t_d was set to 10 seconds, the discount factor γ was set to 0.9, and the size of $|L_{grid}|$ was set to 400×400. The USV and target boat were both given a maximum surge speed of 0.4 m/s. The value of r_{min} was fixed to 1 m, while r_{max} was varied between 2 m and 6 m. The lengths of the target boat and USV were both 0.7 m. Finally, the lag time used by the heuristic motion goal selection algorithm t_{lag} was set to $(r_{min} + r_{max})/(2 \cdot v_{max})$, encouraging the USV to position itself half way between r_{min} and r_{max}.

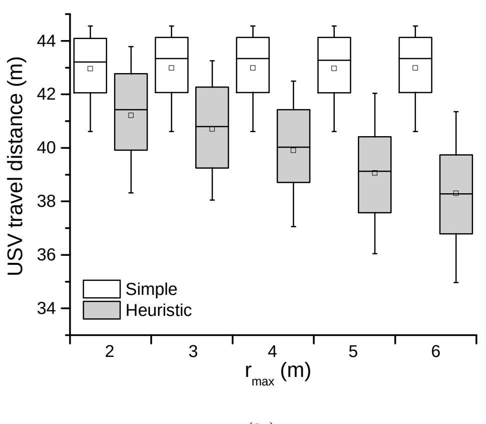
4.4.2 Performance Comparison

A series of experiments were performed across each of the test cases described in Sec. 4.4 as the radius of r_{max} varied between 2 m and 6 m. The results from each experiments are presented in Fig. 4.5, with a more detailed breakdown in Fig. 4.4 for cases where the obstacle density is fixed at 96 obstacles.

The results in Fig. 4.4 show that USVs using heuristic motion goal selection spent a larger portion of their time in the proximity region X_P when compared to USVs using the simple strategy which used the current state of the target boat as a motion goal. When r_{max} was set to 2 m, USVs using the heuristic strategy remained within X_P a total of 10% more often than USVs using the simple strategy. However, when r_{max} increased to 5 m, both strategies were able to stay within X_P for almost the entire duration, eliminating any significant difference between the two.

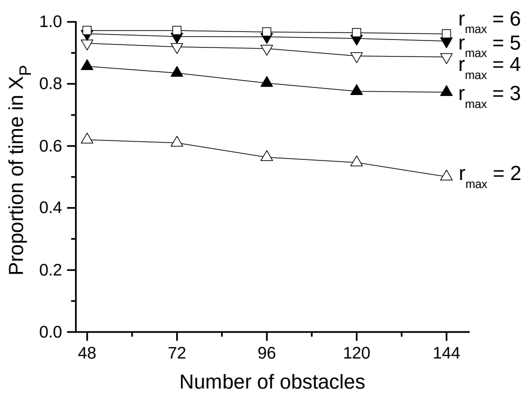


(a)

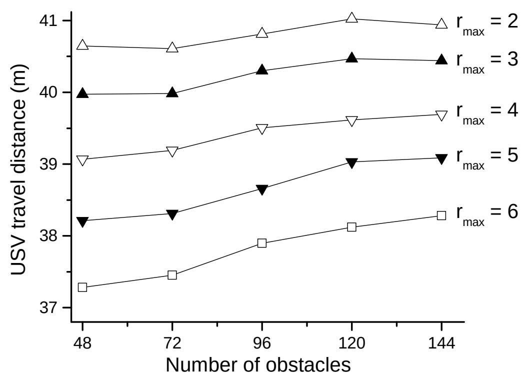


(b)

Figure 4.4: Comparison of simple and heuristic motion goal selection for a set of 200 randomly generated test cases with 96 obstacles.(a) Portion of time the USV spends in the proximity region. (b) Average distance traveled by the USV. For each figure, the box plot shows the median, upper and lower quartile. The whiskers show the upper and lower decile. The mean value is represented by a small square.



(a)



(b)

Figure 4.5: USV performance while using heuristic motion goal selection for various obstacle densities and r_{max} values. (a) Proportion of time the USV spends in the proximity region. (b) Average distance traveled by the USV.

Increasing r_{max} also caused a reduction in the travel distance for USVs using heuristic motion goal selection. When r_{max} increased from 2 m to 6 m, there was approximately a 10% reduction in path length. This is as expected, since as r_{max} increases, less work needs to be done to keep the USV inside proximity region X_P. Fig. 4.2b illustrates the reduction in path length caused by switching to the heuristic motion goal selection, with an approximately 2 m shorter path than the one generated by the USV using the simple tracking in Fig. 4.2a.

As the obstacle density increased, the average time spent in the proximity region decreased for USVs using both heuristic motion goal selection and motion goals determined by the simple strategy. However, in all cases, the heuristic motion goal selection either performed better than the simple strategy, or obtained equivalent performance. There was little noticeable difference between changes in obstacle density when r_{max} was greater than 5 m.

Although the simulation was not performed in real-time, the heuristic motion goal selection algorithm had an average running time of 1.78 s, with a standard deviation of 1.07 s, just slightly slower than the allotted time of 1 s to perform the computation. The computation in real-time could be achieved by utilizing the dimensionality reduction for the path planner as described by Švec et al. [55] and optimization of the planner software.

4.5 Discussion

This chapter introduced a heuristic motion goal selection algorithm enabling an unmanned sea surface vehicle (USV) to safely follow a dynamically moving target through an obstacle field. The algorithm uses Monte Carlo sampling and model based simulation to estimate the future pose of a target boat, and then selects candidate motion goals based on a heuristic cost function. The heuristic tightly integrates with the path planning algorithm by Švec et al. which generates dynamically feasible paths satisfying the differential motion constraints of the USV [55].

The performance of the heuristic motion goal selection algorithm was evaluated through a series of experiments on randomly generated obstacle fields, and its performance was compared against a simpler approach which used the current location of the target boat as a motion goal. Experimental results showed that USVs using the heuristic algorithm were able to stay in the desired range of the target boat for a larger percentage of time, while simultaneously reducing energy expenditure in terms of distance traveled.

4.5.1 Limitations and Future Work

One clear limitation of the approach introduced in this chapter is that it does not incorporate the learned opponent model introduced in Ch. 3. Before the learned model can be incorporated, it is necessary to extend potential fields to domains with differential motion constraints. While this should be feasible using the methods prescribed in Sec. 3.4.1, it is beyond the scope of this dissertation. Future work could combine this with a number of features from Ch. 2 and Ch. 3, such as limited sensor ranges and multiple pursuers

Another limitation of this work is that the motion goal selection algorithm defined in Sec. 4.3 only selects candidate motions goals from the most likely pose of the target boat at each time step. If the distribution is multi-modal, then this is likely suboptimal, since the ideal location for the USV may be between the two modes. Future work should evaluate broader sampling methods which draw from a larger pool of candidates and evaluate more samples.

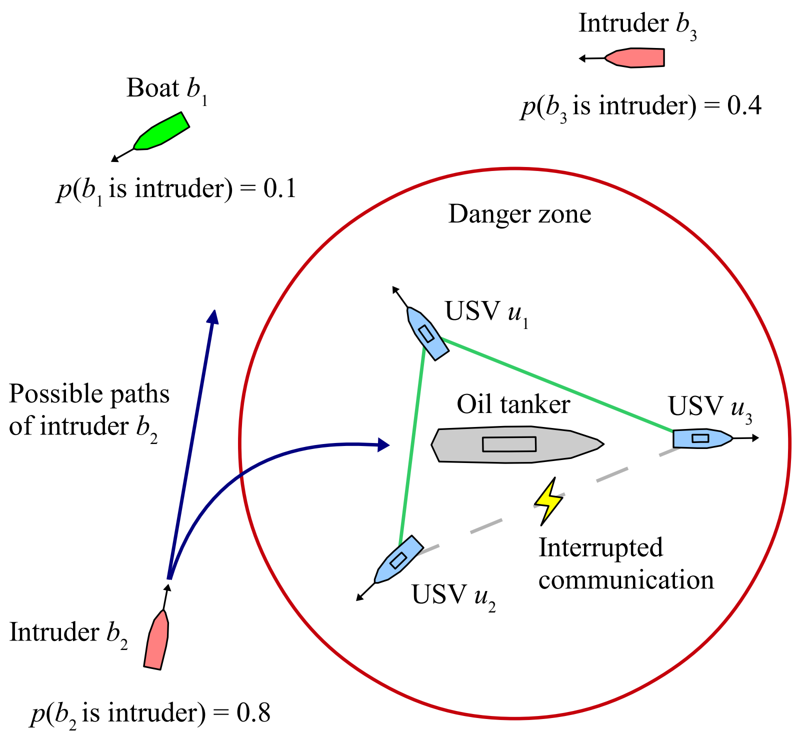
Future experimental evaluation of the approach in this chapter should include a variety of different behaviors from the target boat, such as adversarial motion models, or navigation in environments with dynamic obstacles. This more comprehensive evaluation, combined with the learned motion model from Ch. 3, should give a better perspective on the potential benefits of predictive modeling.

The main contribution of this chapter is that it demonstrates how Monte Carlo sampling techniques can be combined with a black box path planning algorithms to produce a viable control strategy. In the following chapter, we extend this idea further, by demonstrating how a multi-agent task allocation algorithm can utilize model based simulation in a way that integrates planning methods at both high and low levels of abstraction.

Chapter 5: Distributed Asset Guarding

In high-risk marine environments, teams of cooperative, highly-maneuverable unmanned sea-surface vehicles (USVs) can be deployed to guard high-valued assets such as oil tankers or commercial cargo ships from incursion by hostile adversaries. The use of autonomous robotic systems brings several advantages which include reducing the risk of human fatalities and significantly decreasing the cost of missions, while preserving the expected level of security. This, however, imposes multiple challenging requirements on the decision-making capability of these vehicles.

Successfully guarding an asset using a team of USVs requires the cooperative observation of passing boats, identification of potential hostile intruders, and the delay of their progress towards the asset via interception and active blocking [62,63]. Intelligent, balanced decisions about which tasks to perform must be made by the vehicles to keep adversaries away from the asset for as long as possible. This presents a non-trivial challenge for the USVs, since the identity of the hostile boats may not be known in advance. In addition, the possibility of intermittent communication interruptions, noisy sensor data, and the differential constraints of the vehicles all impose additional complications. Finally, the planning approach must be efficient despite the very large state-action space, and should be general enough to be usable



Intruder b_3
$p(b_3$ is intruder$) = 0.4$
Boat b_1
$p(b_1$ is intruder$) = 0.1$
Danger zone
USV u_1
Possible paths of intruder b_2
Oil tanker
USV u_3
Interrupted communication
USV u_2
Intruder b_2
$p(b_2$ is intruder$) = 0.8$

Figure 5.1: A team of unmanned sea-surface vehicles (USVs) guard an oil tanker from hostile intruder boats. During the operation, each boat is assigned a probability of being hostile based on observations made by the USVs.

for a range of scenarios and missions.

This chapter presents a decentralized, contract-based planning approach where individual USVs perform incremental task exchanges to agree upon on the allocation of high-level tasks. The approach makes use of model-predictive simulation to evaluate of candidate task allocations by projecting the future state of the boats in the scene. Once tasks are assigned, they are realized by corresponding low-level behaviors which have been optimized for the specific missions. These behaviors implement a local, reactive obstacle avoidance and interception strategy that respects the differential constraints of the vehicles.

An experimental evaluation of the algorithm demonstrates that the use of model-predictive simulation leads to significantly higher performance and robustness than task-tailored heuristic rules. Additionally, results show that even using a small sample size, Monte-Carlo sampling is beneficial for dealing with sensor uncertainty or uncertainty about the opponent model. The running time analysis reveals that the algorithm has a fast execution time and low-order polynomial time complexity in relation to the number of USVs, suggesting that the algorithm has the computational efficiency needed for online planning.

This chapter also provides a detailed analysis of the developed approach and discuss lessons learned when designing the algorithm so that this information may be used by robotic practitioners attempting solve similar problems. These contributions include: 1) an analysis of the tradeoff between computational effort and plan quality when varying parameters of the algorithm, 2) an analysis of the scalability/computational complexity of the approach for different numbers of USVs or boats, 3) an analysis of the effect of the errors in model-predictive simulation or opponent model on the planner, 4) an analysis of the impact of sensor noise or communication uncertainty on the planner.

5.1 Background

The problem outlined in this chapter can be decomposed into multiple components, e.g., accelerated simulation [64, 65], trajectory planning for collision-free guidance [53, 55, 65], learning of interception behaviors [62], and multi-agent task allocation and planning. In this chapter, our focus is mostly on task allocation and planning. Hence, this section provide an overview of representative approaches in this domain for intentionally cooperative systems of robotic agents [66]. In particular, we this section focuses on distributed and hybrid teams of agents [67].

A taxonomy of multi-robot task allocation (MRTA) problems is provided by Gerkey et al. [68]. Relevant features include the number of tasks that can be performed by a single robot (i.e., ST as single-task robots, and MT as multi-task robots), the number of robots that may be required to fulfil a task (i.e., SR as single-robot tasks, and MR as multi-robot tasks), whether the current assignment of tasks is optimized for future task assignments or not (i.e., IA as instantaneous task allocation and TA as time-extended allocation), and the level of task interdependencies. Our work mostly falls into the MT-MR-TA category, for which problems are generally $\mathcal{NP}$-hard, suggesting that efficient computation of the optimal task allocation is infeasible.

Techniques for solving MRTA problems can be categorized into behavior-based and negotiation-based approaches [66, 69] depending on whether the robots solely rely on the states of other robots and their capabilities, or explicitly communicate to decide on the tasks. The behavior-based approaches do not favor explicit communication among agents to allocate the tasks. Rather, the task allocation is decided based on the known states and skills of other agents in a purely distributed manner. A short survey of the techniques that belong to this category can be found in [66].

Our task allocation algorithm, an extension of our previous work [63], belongs to the category of negotiation-based approaches where agents must individually communicate to decide on the task assignment. This negotiation-based category includes contract and market-based techniques for allocating tasks between agents, as opposed to more centralized approaches [70]. A survey on the current state-of-the-art market-based techniques for multirobot task allocation is given in [66,67,71].

Most of the currently existing market-based approaches for cooperative task assignment are based on the Contract Net Protocol (CNP) [72], one of the pioneering negotiation (auction) protocols for implementation of task allocation algorithms in a distributed setting. According to this protocol, the robots explicitly communicate and negotiate tasks using a specific strategy. This leads to a gradual improvement in the assignment of tasks to robots that have the best capabilities to perform them. The market-based task allocation approaches differ on the type of the negotiation protocol. The protocol defines the way in which the agents offer or request tasks given their capabilities, how many of these tasks can be involved in a single contract (e.g., cluster contracts if more than one task is dealt with), how many agents are involved in a single negotiation (e.g., so-called multi-agent contracts if more than two agents are considered), or whether the agents offer tasks in exchange for another task, e.g., in the form of swap contracts in exchange-like auctions [71, 73].

A distributed, market-based approach MURDOCH for hierarchical task allocation was introduced in [74]. The approach is based on the CNP protocol and publish/subscribe communication model. It can handle robot failures by reassigning the tasks to the most suitable robots in a greedy fashion, and can consider newly created tasks in the allocation process.

The TraderBots [75] market-based approach was developed for distributed coordination of self-interested agents. The approach is known for its capability to create centralized sub-groups within the distributed team to improve the global task allocation efficiency. The approach is able to deal with disruptions in communication through exchange task style of auctioning. Zlot and Stentz present an extension of the TraderBots approach for complex task allocation in [76]. The approach explicitly considers the task structure and its properties to produce more efficient allocations. This includes complex decisions on subtasks sharing among the robots. The subtasks are hierarchically arranged into a task tree, which allows them to be negotiated at different levels.

The Hoplites approach [77] was developed for solving a complex coordination of a distributed group of robots with highly coupled tasks. The approach is market-based is one of the first approaches able to solve the ST-MR-IA type of problems. It provides planned coordination in addition to the tight coordination capability. It combines two different coordination strategies, i.e., the passive, purely local strategy according to which the robots have to quickly decide on the tasks, and the active, market-based strategy which allows them to agree on the tasks in more complex scenarios.

A task allocation approach for computing a combination of strongly and weakly cooperative solutions for a group of heterogeneous robots operating in the context of a single task allocation application was introduced in [78]. More specifically, the ASyMTRe-D algorithm for the synthesis of coalitions within the group was combined with a market-based approach for the allocation of weakly cooperative tasks. According to the authors, the approach is thus highly flexible and amenable to a large variety of robotic applications.

A game-theoretic approach is introduced in [79] for solving the “Mobile Resources protecting Moving Targets” (MRMT) problem, where multiple defenders must guard a set of moving targets against multiple attackers. The authors represent the problem as a continuous time Stackelberg game and use linear programming to find a Strong Stackelberg equilibrium. This approach has the advantage of offering a minimum performance guarantee against any possible opponent, something that our work lacks. However, compared to our work, the interaction between defenders and attackers in their model is highly simplistic. They do not consider the movement of the attackers, so the notion of blocking or intercepting an attacker is only dealt with abstractly. The attacker may choose an arbitrary time to attack one of the targets, and the probability of the attack succeeding is determined by whether or not the target is currently within the “protection radius” of one or more defenders. In contrast, the work in this chapter explicitly models the movement of USVs and intruders when blocking or intercepting, and includes that information directly in the strategy evaluation. This work also addresses the problem of neutral, non-hostile boats that must be distinguished from intruders through observation, something which previous work does not directly address.

Similar to the work above, the work in [80] computes a Stackelberg equilibrium for a defensive game with multiple moving targets and multiple attackers. Unlike the work above, this work does consider the movement of the attackers, but the problem space is very heavily discretized. Their approach depends on solving an $\mathcal{NP}$-hard non-linear program, so the largest problem they evaluated used a 5x4 grid to represent the environment. Because their model is fairly coarse, this work does not address differential constraints or realistic blocking behavior. This approach was utilized in the marine domain to protect merchant ships against pirate boats [81]. Additional work on Stackelberg games in this domain includes the incorporation of Quantal Response models for adversary behavior [82] and accounting for the constrained mobility and limited endurance of defender agents [83].

In the USV domain, a decentralized, behavior-based STAGS approach for a multi-USV system to protect sensitive areas against intruders was developed in [84]. The deployment of the vehicles is controlled by a heuristic algorithm that uses dynamically created gaps between the vehicles and the asset. The parameters of the approach are optimized to improve its performance by minimizing the average response time and missing rate.

Purely rule-based approaches include [85] as a part of the Swarm Management Unit (SMU) used for controlling a team of USVs to carry out surveillance and guarding an asset by intercepting detected intruders. The approach selects which USVs should intercept a detected intruder based on domain-specific heuristics. The positions of the USVs are optimized according to two criteria, i.e., preventing the intruders from getting too close to the asset and minimizing the interception time. The aim is to find a balance between the desired coverage of the area around the asset and the level of security.

Related research also includes techniques for patrolling a polygonal area using a group of agents. A survey of the current state-of-the-art patrolling algorithms is provided in [86]. The representative approaches are evaluated in detail in [87] in terms of the average idleness of a patrolling graph and scalability to the number of agents metrics. In our approach, the patrolling strategy is computed indirectly through the market-based exchange of guard tasks commanding the vehicles to computed waypoints or predefined patrolling locations.

In contrast to prior approaches, our work makes explicit consideration of sensing uncertainty when differentiating intruder boats from other non-hostile boats (i.e., by requiring observation of passing boats to identify threats), and accounts for differential constraints of the USVs and their complex interaction with the intruders when allocating tasks (i.e., when executing intercepting and blocking strategies). We show that by carefully integrating the model-predictive simulation with the underlying task allocation, features such as differential constraints and sensing uncertainty can be directly considered during task allocation and still run efficiently. We provide a detailed analysis of the developed approach and describe lessons learned for realizing high-fidelity task allocation with unmanned surface vehicles.

5.2 Problem Formulation

This chapter defines a multi-agent planning problem where a team of USVs must defend a stationary target from an attack by a set of hostile intruder boats interspersed among other non-hostile boats. An example of this scenario is depicted in Fig. 5.1. The USV team does not know *a priori* which boats are hostile intruders, but can estimate the probability that a boat is an intruder through observation. Once an intruder is identified, an alert is triggered, and the objective of the USV team becomes to delay the hostile boats from reaching the target for as long as possible. This is done by actively blocking the path of the intruders, forcing the intruders to slow down or change direction.

In addition to uncertainty about which boats are intruders, the USV team must deal with noisy sensor data, which creates uncertainty over the position of passing boats, and random communication interruptions, which create periods of time where USVs cannot exchange information directly.

The motion model used by the USVs and boats is the same as was described in Chapter 4, Sec. 4.2. The formal definition of the problem also includes:

(i.) a team of USVs $U = \{u_1, u_2, \dots u_m\}$ responsible for defending the target from intruder boats,

(ii.) a set of passing boats $B = \{b_1, b_2, \dots b_n\}$ including a subset of one or more hostile intruders $I \subseteq B$,

(iii.) the location l_{target} of the stationary target,

(iv.) a non-finite set of observations Ω, where each observation $o_{b_j} = \langle \widetilde{x}_{b_j}, f_{b_j} \rangle$ provides a noisy estimate $\widetilde{x}_{b_j}$ of the vehicle state x_{b_j} and observed features f_{b_j} of boat b_j (e.g. color, size, etc.),

(v.) a pair of opponent models, $M_B(x_{i,t+k}|x_{i,t}, O_t)$ and $M_I(x_{i,t+k}|x_{i,t}, O_t)$, for passing boats and intruders respectively, which define the probability that boat b_i will transition from state $x_{i,t}$ to $x_{i,t+k}$ given observation history O_t,

(vi.) an intruder classification function $P(b_i \in I|O_t)$ which returns the probability that boat b_i is an intruder given observation history O_t,

(vii.) a response team probability threshold p_{alert}, which defines the probability $P(b_i \in I|O_t)$ above which an alert will be triggered.

The policy π_{u_i} of USV u_i provides a mapping from observation histories to the control actions $q \in Q(x_{u_i})$. The objective of the USV team is to find a set of policies $\Pi_U = \{\pi_{u_1}, \pi_{u_2}, \ldots, \pi_{u_m}\}$ that maximize the expected delay time, $E[t_{delay}]$, defined as the time difference between when the alert is first triggered and when an intruder first arrives at the target, or $t_{delay} = t_{arrival} - t_{alert}$. Thus, the optimal set of policies Π_U^* is defined as,

$$\Pi_U^* = \arg\max_{\Pi_U} E[t_{delay}|\Pi_U]. \tag{5.1}$$

Exactly computing Π_U^* is likely to be intractable, so we are interested only in finding a set of policies that can be computed efficiently, even if they are sub-optimal.

The motivation behind maximizing $E[t_{delay}]$ is to provide a hypothetical response team as much time as possible to deal with the intruders. Although the response team is not modeled explicitly as part of the problem, I assert that the longer intruders are delayed from reaching the target, the more time the response team will have to repel an attack, do emergency preparations, or perform other task which are beneficial the overall mission. Additionally, by having the alert triggered by the response team, the USVs is isolated from the responsibility of classifying intruders or weighing the cost of false alarms.

To accurately reflect the limitation of USVs' knowledge, policy π_{u_i} must depend only on information that is accessible to USV u_i at the time the policy is executed. The information known to USV u_i at time t, includes:

(a.) the state x_{u_i} of USV u_i up to time t

(b.) USV u_i's observation history O_{u_i} up to time t

(c.) information received from other USVs before time t

When in communication, USVs are permitted to exchange arbitrary information, including their current state and observation histories. USVs are also assumed to have access to opponent models M_B and M_I, classifier $P(b_j \in I|O_{u_i})$, target location l_{target} and response team threshold p_{alert} when computing their policies.

The parameterization of opponent models M_B and M_I, and intruder classification function $P(b_i \in I|O_{u_j})$ are described in the experimental setup in Sec. 5.4. The noisy state estimates $\widetilde{x}_{b_j}$ produced via observation are assumed to be the result of adding two-dimensional Gaussian noise to the components of state vector x, with variances also provided in Sec. 5.4.5.

5.3 Approach

This section presents a solution technique for the task assignment problem described in Sec. 5.2. Our approach, illustrated in Fig. 5.2, consists of three major components: 1) a decentralized task allocation process that determines the high-level task assignment of each USV, 2) a set of parameterized behaviors that map each

USV's task assignment into a unique motion goal, 3) a control action policy that selects a control action for each USV to reach its motion goal while performing local obstacle avoidance. Each of these processes operate concurrently and are performed online, updating as new sensor information becomes available.

Our approach utilizes a set of domain-specific tasks that may be specified by the system developer. For the purposes of this chapter, we define the set of high-level tasks, $H \subseteq H_o \cup H_g \cup H_d$, as the union of three distinct task types:

(i.) a set of *observe* tasks, H_o, where USVs are responsible for approaching and gathering information about passing boats

(ii.) a set of *guard* tasks, H_g, where USVs must position themselves at predetermined locations around the target

(iii.) a set of *delay* tasks, H_d, where USVs must intercept and block a hostile boat

Each task $h \in H_o \cup H_d$ specifies a single boat to observe or delay, while each $h \in H_g$ specifies a single location to guard. The task assignment $H_{u_i} \subseteq H$ for USV u_i may contain any combination of these tasks.

The joint task allocation $\mathcal{A} = \{H_{u_1}, H_{u_2}, \ldots H_{u_m}\}$ defines the current task assignment for all USVs and is computed online and updated during each planning step via a decentralized task allocation process. Our algorithm uses both heuristic and model-predictive simulation to determine how the task allocation should be updated. This process is described in Sec. 5.3.1.

Since communication between USVs can be interrupted, each USV u_i maintains a local task allocation $\mathcal{A}_{u_i}$ representing u_i's belief about the joint task alloca-

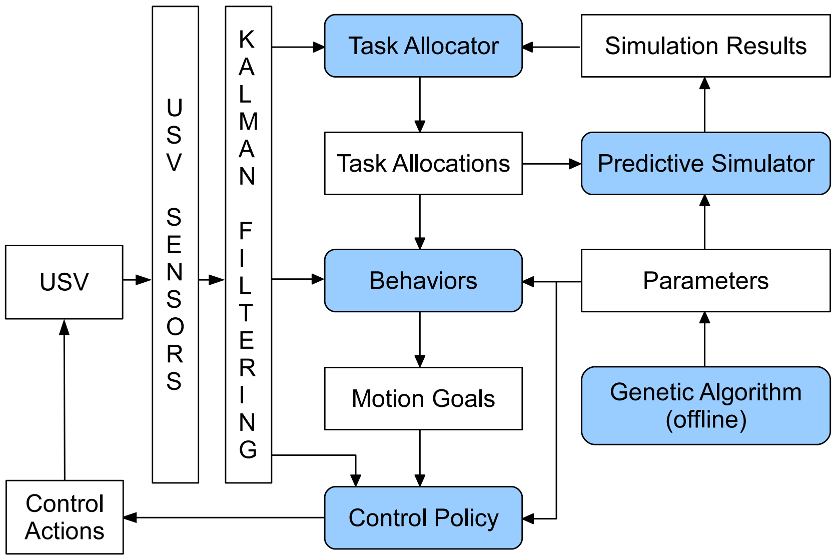


Figure 5.2: Planning and control architecture for one USV.

tion $\mathcal{A}$ based on the most recent information made available to u_i. The local task allocation $\mathcal{A}_{u_i}$, together with the observation history O_{u_i}, form the input to USV u_i's motion goal selection $G_{u_i}(O_{u_i}, \mathcal{A}_{u_i})$ and control action selection $\pi_{u_i}(O_{u_i}, \mathcal{A}_{u_i})$ policies. These are defined in Appendix, Secs. A.1 and A.2.

Both motion goal selection and control action selection policies are based on hand-coded heuristics which are tuned offline using a genetic algorithm. A set of tuning parameters $\Gamma = \{\gamma_{guard}, \gamma_{intr}, \gamma_{dist}, \gamma_{lead}, \gamma_{block}, \gamma_{max}, \gamma_{initial_r}, \gamma_{occupied}, \gamma_{slow_r}, \gamma_{goal_r}, \gamma_{fan_r}, \gamma_{fan_\theta}\}$ determine the low-level behavior of the USVs when selecting motion goals or performing obstacle avoidance. The tuning process used for these parameters is described in Sec. 5.3.5.

The sequence of state estimates $\{\widetilde{x}_{b_j,1}, \widetilde{x}_{b_j,2}, \ldots \widetilde{x}_{b_j,n}\}$ for boat b_j contained in USV u_i's observation history O_{u_i} are processed using a simple Kalman filter, where $l_{b_j}(O_{u_i})$ represents the estimate of boat b_j's location given O_{u_i} and $K_{b_j}(O_{u_i})$

represents the corresponding covariance matrix. We will refer to these as simply l_{b_j} and K_{b_j} throughout the chapter.

5.3.1 Task Allocation

The joint task allocation $\mathcal{A}$ is periodically updated via a decentralized task re-allocation process that is performed concurrently by each of the USVs. This process behaves like a local hill-climbing algorithm, where each USV u_i evaluates variations of the task allocation $\mathcal{A}_{u_i}$ that differ by exchanging one or two tasks. This process is less computationally demanding than evaluating all possible task allocations. It also makes it easier to decentralize the re-allocation process, since each exchange alters the task assignment of at most two USVs. However, a disadvantage of this approach is that it will only find locally optimal solutions, a limitation common to many hill-climbing algorithms.

Before the task re-allocation process can be performed, an initial task allocation $\mathcal{A}_{u_i,0}$ must be assigned to the USV team. We assume that the initial allocation will be assigned by the system developer. In our case, we assign each USV one or more guard tasks $h_j \in H_g$, distributed uniformly at radius $\gamma_{initial_r}$ around the target. For each new boat b_j that enter the scene, the nearest USV assigns itself an observation task $h_j \in H_o$. If any boats are identified as intruders, meaning $P(b_i \in I|O_{u_j}) > p_{alert}$, the observation task for that boat becomes an equivalent delay task, $h_j \in H_d$.

At regular time interval t_{alloc}, each USV u_i performs a task re-allocation step,

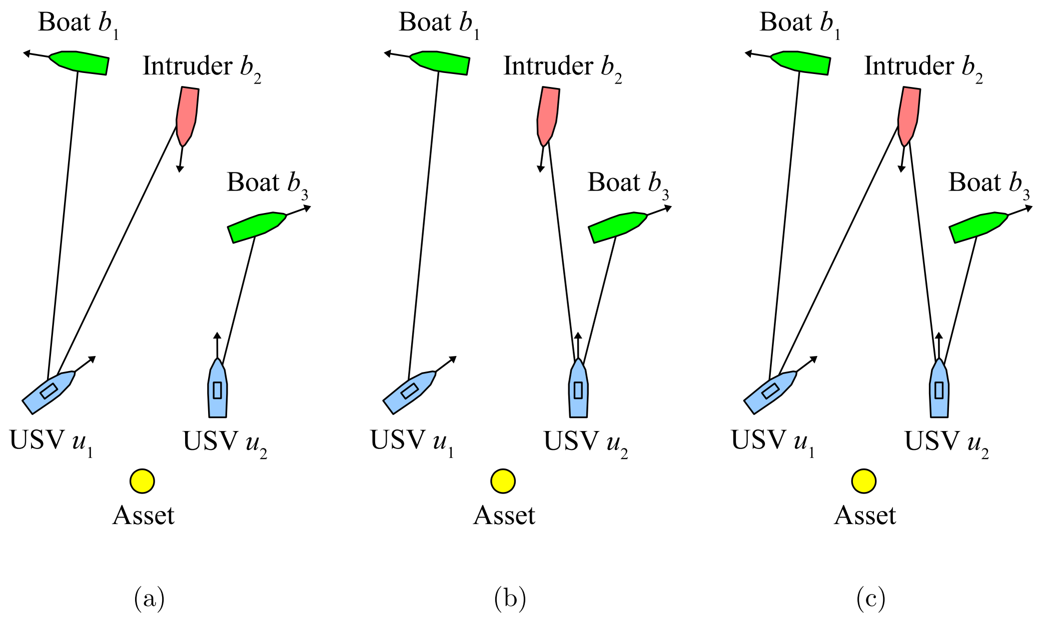


Figure 5.3: Candidate task allocations a) the current task allocation $\mathcal{A}$ without modification, b) modification of $\mathcal{A}$ with a delay task offered to USV u_2 by USV u_1, c) modification of $\mathcal{A}$ with a delay task shared to USV u_2.

in which u_i computes a revised allocation $\mathcal{A}'_{u_i}$, defined as

$$\mathcal{A}'_{u_i} = \arg\max_{\mathcal{A}_{u_i,j} \in C} \widetilde{E}[t_{delay}|O_{u_i}, \mathcal{A}_{u_i,j}] \tag{5.2}$$

where C is a set of candidate task allocations produced by Alg. 4, and $\widetilde{E}[t_{delay}|O_{u_i}, \mathcal{A}_{u_i,j}]$ is the estimated utility of candidate $\mathcal{A}_{u_i,j}$ given observation history O_{u_i} as computed by the model-predictive simulation described in Sec. 5.3.3. Each candidate $\mathcal{A}_{u_i,j} \in C$ differs from $\mathcal{A}_{u_i}$ by sharing or offering one or more tasks from H_{u_i} to another USV, as depicted in Fig. 5.3 and 5.4.

The method GenerateCandidates$(O_{u_i}, \mathcal{A}_{u_i}, u_i)$ in Alg. 4 produces a set of candidate task allocations C by iterating through every task $h_j \in H_{u_i}$ in USV u_i's task assignment and setting up potential exchanges with every USV $u_k \in U$ on the

Algorithm 4 GenerateCandidates$(O_{u_i}, \mathcal{A}_{u_i}, u_i)$

```
Require: The observation history $O_{u_i}$ and task allocation $\mathcal{A}_{u_i}$ for USV $u_i$
Ensure: A set of candidate task allocations $C$.
 1: $C \leftarrow \{\mathcal{A}_{u_i}\}$
 2: for each $h_j \in H_{u_i} \cap H_d$ do
 3:     for each $u_k \in U$ do
 4:         $C \leftarrow C \cup$ ShareTask$(\mathcal{A}_{u_i}, h_j, u_k)$
 5:     end for
 6: end for
 7: for each $h_j \in H_{u_i} \cap (H_g \cup H_o)$ do
 8:     for each $u_k \in U$ do
 9:         $C \leftarrow C \cup$ OfferTask$(\mathcal{A}_{u_i}, h_j, u_i, u_k)$
10:     end for
11: end for
12: return $C$
```

team. These include both share and offer exchanges, where a task is either passed exclusively to USV u_k, or duplicated and shared with u_k.

The method ShareTask$(\mathcal{A}_{u_i}, h_j, u_k)$ returns a new task allocation $\mathcal{A}'_{u_i}$ that differs from $\mathcal{A}_{u_i}$ by appending task h_j to USV u_k's task assignment H_{u_k}. The result is a task allocation where both u_i and u_k share task h_j.

The method OfferTask$(\mathcal{A}_{u_i}, h_j, u_i, u_k)$ is very similar to ShareTask, only it removes task h_j from USV u_i's task assignment H_{u_i} before adding it to USV u_k's task assignment H_{u_k}. The method can also be modified to include conditional "swap" exchanges, composed of two distinct offer exchanges, depicted in Fig. 5.4. This is applicable to situations where USV u_i is already assigned a delay task h_1 and then USV u_j offers u_i an additional delay task h_2. USV u_i cannot intercept and block two different boats at the same time, so u_i will "swap" task h_1 for task h_2. This is equivalent to u_j offering to h_2 to u_i and u_i offering to h_1 to u_j, but both exchanges are represented by a single candidate task allocation, allowing the

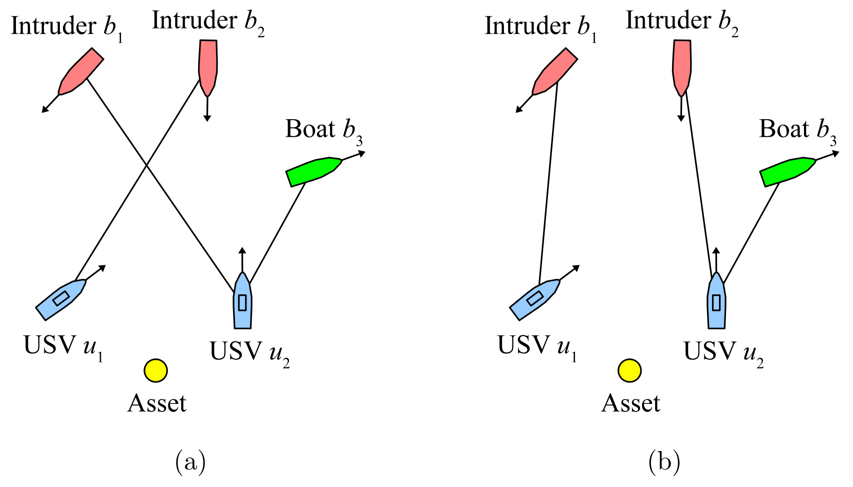


Figure 5.4: A conditional "swap" exchange, composed of two offer exchanges a) the current task allocation $\mathcal{A}$ where USV u_2 by USV u_1 are assigned delay tasks for separate intruders, b) modification of $\mathcal{A}$ where USV u_2 by USV u_1 have swapped delay tasks.

combined exchanges to be evaluated by the predictive simulation.

5.3.2 Communication Protocol

Communication between USVs is modeled as a network of pairwise connections, as shown in Fig. 5.5. At each time step t_k, each USV attempts to synchronize its information with the other USVs. If USV u_j is able to communicate with USV u_i, then u_j's most recent observation history O_{u_j} and task assignment H_{u_j} of USV u_j are merged into USV u_i's observation history O_{u_i} and global task allocation $\mathcal{A}_{u_i}$. If USV u_i cannot directly communicate with USV u_j, it may still learn of u_j's observations and task assignment through a third USV that can communicate with both agents, but it will take an additional time step for this information to propagate.

For USV u_i to either offer or share a task with another USV u_j, the two USVs must explicitly agree to the exchange. If no communication is possible between u_i

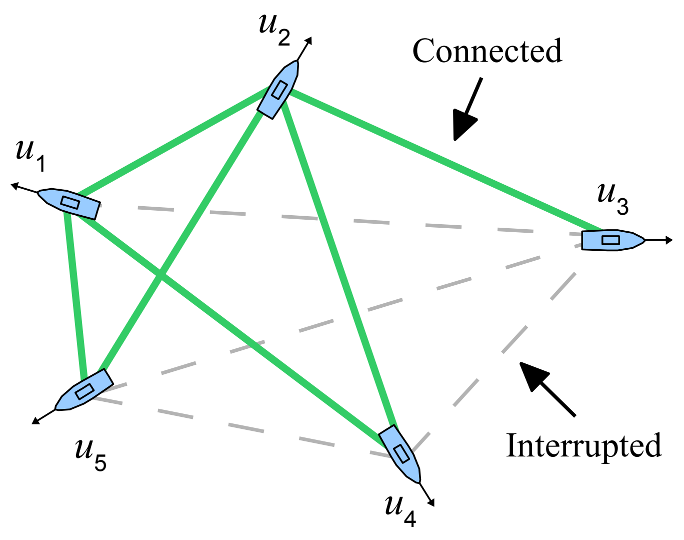


Figure 5.5: An example of a communication network.

and u_j, then exchanges between these agents cannot be performed until communication is re-established. Since communication interruptions occur at random, the USV evaluates all candidate exchanges even if communication is not possible, then determines whether communication has been restored once the evaluation is finished. If the best-performing candidate allocation cannot be applied due to interrupted communication, then the USV selects the next best candidate.

To avoid conflicts that might result from concurrent exchanges, the protocol requires that USV u_i retain the exclusive right to assign tasks from H_{u_i} to another USV. For offer and share exchanges, this rule is trivial to enforce, since USV u_i is the agent that initiates the exchange. However, for "swap" exchanges, USV u_i must ask another USV u_j to offer some task h_2 in exchange for the task h_1. Since exchanges are evaluated concurrently, task h_2 may no longer be in u_j's task assignment at the time the request is made. In this situation, USV u_i must forego the swap and perform a different exchange from its list of candidates.

5.3.3 Predictive Simulation

During task re-allocation process, USV u_i uses model-predictive simulation to evaluate the set of candidate task allocations C generated by the method in Alg. 4. The predictive simulation estimates the expected delay time $E[t_{delay}|O_{u_i}, \mathcal{A}_i]$ for each candidate task allocation $\mathcal{A}_i \in C$, given USV u_i's observation history O_{u_i}. The simulation uses the probabilistic opponent models defined in Sec. 5.2 to estimate the future control actions of passing boats and intruders, and uses the control action selection policies for the USVs defined in Sec. A.2 to estimate the future control actions of other USVs.

USV u_i estimates $\mathcal{A}_i$'s performance for the set of possible worlds W consistent with USV u_i's observation history O_{u_i}. Since USV u_i is uncertain which boats are intruders and about the location of each boat, each possible world $w_j \in W$ consists of a set of possible intruder boats $I_j \subseteq B$ and an approximate of the global state $s_j \in S$. The set W is non-finite, so the algorithm uses Monte-Carlo sampling to select n_{sample} possible worlds to estimate the expected utility, $E[t_{delay}|O_{u_i}, \mathcal{A}_i]$, for each candidate task allocation $\mathcal{A}_i \in C$.

For simplicity, we assume the probability distributions over I and s are statistically independent, meaning

$$P(w_j|O_{u_i}) = P(I_j = I|O_{u_i})P(s_j = s|O_{u_i}).$$

where I is the true set of intruders and s is the true global state. Each global state s_j is sampled using the state estimate l_{b_j} and covariance matrix K_{b_j} produced by the Kalman filter described in Sec. 5.3. The algorithm samples the set of intruders I_i with probability,

$$P_{I_i} = \left(\prod_{b_j \in I_i} P(b_j \in I | O_{u_i}) \right) \left(\prod_{b_j \in B \setminus I_i} 1 - P(b_j \in I | O_{u_i}) \right)$$

where P_{I_i} is an approximation of $P(I_i = I | O_{u_i})$ computed by assuming that the appearance of each intruder is statistically independent. In the special case where $n_{sample} = 1$, the possible world w_j with the highest probability $P(w_j | O_{u_i})$ is sampled, rather than a random sample.

The task re-allocation process is not simulated during the predictive simulation. This is to prevent the predictive simulation from recursively calling itself, which would lead to an exponential increase in the computational workload as the simulation searches deeper. Each trial is also given a maximum lookahead time, $t_{lookahead}$, after which the utility is estimated using the intercept point heuristic described in Sec. 5.3.4.

5.3.4 Heuristic Evaluation Function

After the maximum lookahead time $t_{lookahead}$ has expired, the predictive simulation must use a static evaluation function to estimate the value of the resulting state. To do this, we compute intercept point l_{int} defined in Sec. A.1, and estimate the arrival time of an intruder by measuring its straight-line distance from the asset.

The estimated arrival time is delayed by an amount defined in the calculation of l_{int} given how many USVs are currently assigned to delay it and their distance from the intercept point. This estimate is not as accurate as performing a full predictive simulation, but it can be performed very quickly, which is useful for evaluating the state of the world after the maximum lookahead time has passed. When multiple intruders are present, we use the minimum time estimate across all intruders to approximate the expected utility $E[t_{delay}|O_{u_i}, \mathcal{A}_i]$ for the USV team.

USVs can also directly use heuristics to evaluate task exchanges as an alternative to predictive simulation. For example, to calculate the value of exchanging a delay task, performing predictive simulation with $t_{lookahead} = 0$ will provide an efficient albeit less accurate evaluation. When exchanging guard or observe tasks however, the estimated arrival time from the intercept point calculation does not provide useful information, so an alternative heuristic must be defined.

To evaluate the exchange of guard or observe tasks without using predictive simulation, we developed a heuristic rule that prioritizes the exchange of task $h_j \in H_{u_i}$ whose motion goal G_{h_j} is furthest from USV u_i's current motion goal G_{u_i}. This task is offered to another USV u_k whose current motion goal is closest to task h_j's motion goal. If u_k is already assigned a delay task, then the distance to h_j is multiplied by $\gamma_{occupied}$, to discourage giving too many tasks to USVs that are already responsible for delaying an intruder.

Using these pure heuristic rules, we provide an alternative to the predictive simulation which can be used for comparison purposes. The experimental results in Sec. 5.4 show, among other things, that using the predictive simulation provides better task allocations than pure heuristics for this problem.

5.3.5 Optimization of Behaviors

We use a genetic algorithm (GA) [88] to optimize the 12 underlying parameters γ_{guard}, γ_{intr}, γ_{dist}, γ_{lead}, γ_{block}, γ_{max}, $\gamma_{initial_r}$, $\gamma_{occupied}$, γ_{slow_r}, γ_{goal_r}, γ_{fan_r} and γ_{fan_θ}, of the behavior and control action selection policies to further improve the expected utility of the USV policy. The optimization of these parameters allows the USVs to make balanced decisions between guarding a certain location, observing incoming boats, and intercepting and delaying the movement of identified intruders.

For the results presented in Sec. 5.4, the genetic algorithm was run for 150 generations using a population size of 100 chromosomes, where each chromosome represented a complete set of parameters. The parameters for the initial population were assigned at random, while subsequent populations were bred based on the fitness values of the previous generation. Each chromosome's fitness was measured using the median *delay time* from 250 random simulation runs. We utilized roulette wheel selection to determine the breeding population, and applied genetic operators with a crossover rate of 0.35 and mutation rate of 0.08.

5.3.6 Complexity Analysis

Given the set of tasks H and the set of USVs U, the number possible task allocations is $O(|2^{H^U}|)$, which is too large to explore exhaustively. The number candidate task allocations selected by Alg. 4 is a more modest $O(|H \times U|)$. This means the number of candidates evaluated increases linearly as the number of USVs or tasks increases. Since we are using predictive simulation to evaluate each candidate, increasing the number of USVs should also increase the time it takes to perform a simulation. As a result, the time complexity of a single task re-allocation step should be $O(|H \times U| \cdot |U|)$, which grows quadratically as the number of USVs increase. This is confirmed by the experimental results in Sec. 5.4.4.

If we consider an arbitrary sequence of task exchanges of length k, then there are $O(|H \times U|^k)$ such sequences. The number of possible sequences grows exponentially as k increases, meaning it will be prohibitively time consuming to evaluate all such sequences for large values of k. For this reason, we only evaluate exchanges of one or two tasks at a time.

In our implementation in Sec. 5.4.1, we narrow the set of candidate task allocations further by pruning certain types of exchanges. First, we eliminate share exchanges for guard and observe tasks, so only one instance of these tasks will exist at a time. Second, we limit the number of USVs that can simultaneously delay a single intruder, determined by the parameter γ_{max}. Both of these changes reduce the total number of tasks in $\mathcal{A}$ by eliminating redundant assignments while still preserving at least one copy of each task. Additionally, we only consider "swap" exchanges in the case described in Sec. 5.3.1 so that the number of candidates evaluated remains $O(|H \times U|)$.

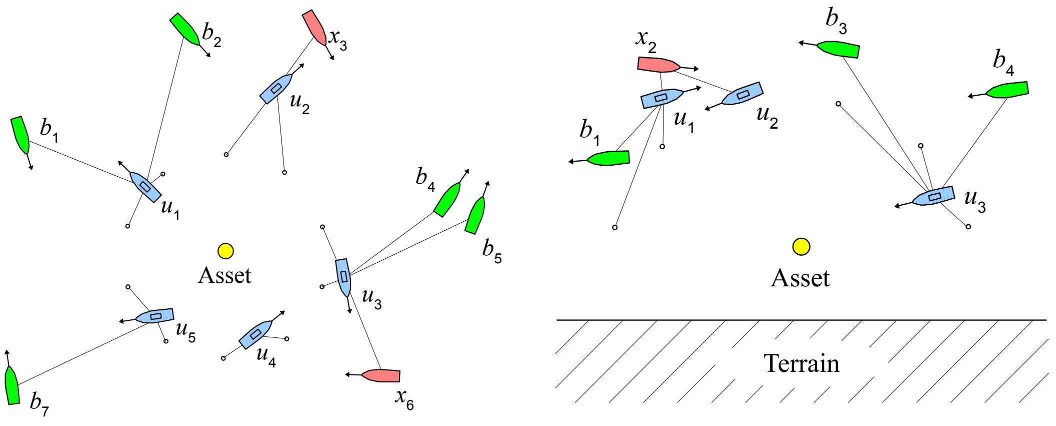
5.4 Experimental Results

We have evaluated our planning approach by performing experiments in two simulated scenarios, depicted in Figs. 5.6a and 5.6b. The motion model used for the experiments is the same motion model used by the predictive simulator, which was defined by Eqn. 4.1 in Sec. 5.2. Details about the parametrization of the simulator are provided in the experimental setup below, followed by results and discussion for a number of different experiments. We also discuss the limitations of these experiments in Sec. 5.5.1.

5.4.1 Experimental Setup

In scenario 1, shown in Fig. 5.6a, the target is positioned within a circular region without any static obstacles. In scenario 2, shown in Fig. 5.6b, the target is positioned near static terrain, restricting the direction of incoming boats. In scenario 1 there are a total of 5 USVs and 3 intruders, while in scenario 2 there are 3 USVs and 2 intruders. In both scenarios, passing boats will continuously enter and leave the operating space, with a maximum of 8 passing boats appearing in the scene at any given time.

At the beginning of each trial, the positions of passing boats are initialized at random locations around the target. During each trial run, new boats appear at random locations along the boundary of the operating space, which is defined as a ring in scenario 1, (with an inner and outer radius of 80 and 100 m), or as two rectangles on the left and right sides of the target in scenario 2 (with a distance of



(a) Scenario 1 (b) Scenario 2

Figure 5.6: Two example scenarios where a team of USVs u_i defend a target in an open ocean with several passing boats. Boats marked x_i are identified as intruders. The tasks assigned to each USV are depicted as lines.

80 m from the target and a width of 20 m). The rate at which new boats appear is balanced with the rate at which other boats leave the operating space, restricted to the maximum of 8 passing boats.

Each boat's initial trajectory is a path tangent to a randomly sized circle (or semi-circle in scenario 2) surrounding the target with minimum and maximum radius of 30 and 60 m. Unlike non-hostile boats, intruders will change their trajectory and turn towards the target when they pass within 60 m If an intruder passes within 5 m of a USV, it will assume it is being blocked and start approaching the target immediately. Whether a boat will be an intruder or not is determined by the amount of time elapsed during the simulation. Only non-intruder boats will appear during the first 30 seconds of the simulation, immediately followed by 2 or 3 intruders depending on the scenario. As a result, a group of intruders will always appear at or around the same time in the simulation.

Both intruders and non-intruder boats use the same reactive obstacle avoidance strategy described in Sec. A.2. For intruders, the parameters γ_{fan_r} and γ_{fan_θ} are set to 10 m and 120° respectively, while for non-intruders they are set to 15 m and 180°. The intruder is given a more aggressive set of parameters allowing it to approach other boats more closely before adjusting its trajectory. For USVs, these parameters are optimized using the genetic algorithm described in Sec. 5.3.5. The maximum surge speed for all USVs and other boats is fixed at 10 m/s.

To make the interactions between USVs and intruders more challenging, intruders will perform evasive actions to avoid being blocked. If the intruder is blocked by another boat and diverted away from the target for some time t, where $t > t_{flip}$, the intruder will turn away from the blocking boat and reverse its direction of movement. For each evasive turn, the value of t_{flip} is selected at random, uniformly between 1 and 3 seconds, to introduce non-deterministic behavior into the intruder strategy. We show that intruders which perform evasive turns are more difficult to defend against in Sec. 5.4.2. However, this model is not guaranteed to be a best-response to the USV team's strategy, and therefore cannot be used to determine the worst-case performance against any theoretical opponent.

The observation classification function simulates the probability that each boat is an intruder based on the quality of the observations made by the USV team. It does this by specifying an observation quality value $\alpha_{b_j} \in [0, 1]$ for boat b_j that is initially set to 0 and increases monotonically while USV u_i is within 50 m of boat b_j. If d represents the distance between u_i and b_j, then α_{b_j} increases at a rate of $\delta_{learn}(1-d/50)$ per second, with the default learning rate $\delta_{learn} = 0.5$. This means it takes at most 5 seconds to obtain an observation quality of $\alpha_{b_j} = 1$ when observing boat b_j from a distance of 30 meters.

The function $P(b_j \in I|O_{u_i})$ returns a prior probability of 0.05 when $\alpha_{b_j} = 0$, indicating that no observations have occurred. The choice of 0.05 is arbitrary, but is meant to represent a small non-zero chance that each boat could be an intruder. As α_{b_j} increases, the probability $P(b_j \in I|O_{u_i})$ converges linearly to 1 or 0 depending on whether or not b_i is actually an intruder. Gaussian noise with standard deviation $0.1(1-\alpha_{b_j})$ is added to the probability function so that the change is non-monotonic. For all simulations, the value p_{alert} for determining whether a boat should be classified as a threat was set to 0.6.

During the predictive simulation, the lookahead time $t_{lookahead}$ is set to 5 s, and the Monte-Carlo sample size $n_{samples}$ is set to 5 for all experiments unless otherwise indicated. These parameters are useful for ensuring the predictive simulation can be executed in real time while still obtaining good performance.

5.4.2 Strategy Comparison

To evaluate the performance of the approach defined in Sec. 5.3, we performed experiments on several strategy variants. All the strategies evaluated use the same motion goal and control action selection policies defined in Secs. A.1 and A.2, but they differ in how task allocation is performed:

(1.) a *predictive* strategy, which uses the complete task allocation strategy described in Sec. 5.3, (2.) a *heuristic* strategy, which does not utilize predictive simulation, but performs task allocation based on the heuristic approach described in Sec. 5.3.4,

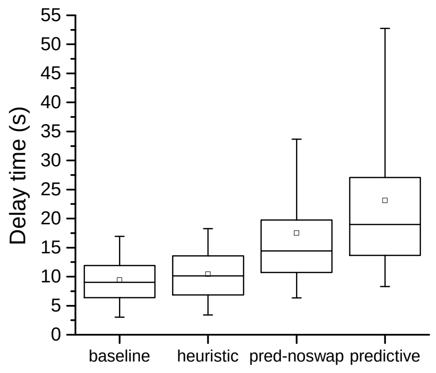
(3.) a *baseline* strategy, which does not perform task exchanges at all, instead each USV waits at its guard location until an intruder is identified, then delay tasks are assigned using the heuristic in Sec. 5.3.4,

(4.) a *pred-noswap* strategy, a variant of the predictive strategy that does not use the conditional "swap" exchange discussed in Sec. 5.3.1.

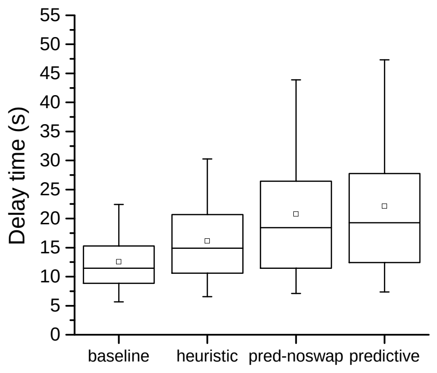
Figures 5.7 a) and b) show the average *delay time* across 1000 randomly generated trial for each of the four different strategies. The box plots show the median, upper and lower quartile of the data set, while the whiskers mark the 5th and 95th percentiles. The mean value is marked with a small square in each figure.

As expected, the *predictive* and *pred-noswap* strategies performed best, followed by the *heuristic* strategy, while the *baseline* strategy performed worst. For scenario 1, the predictive strategy increased the median *delay time* by 110% compared to the baseline strategy and by 87% compared to the heuristic strategy. For scenario 2, the increase was 68% compared to the baseline strategy and 29% compared to the heuristic strategy. The predictive strategy also performed better when the "swap" exchange was included, increasing *delay time* by 31% in scenario 1 and 4.5% in scenario 2.

The difference in *delay time* between the heuristic and predictive strategies is less for scenario 2 when compared to scenario 1, possibly due to the smaller number of choices during the task allocation step, decreasing the likelihood of the heuristic



(a) Scenario 1



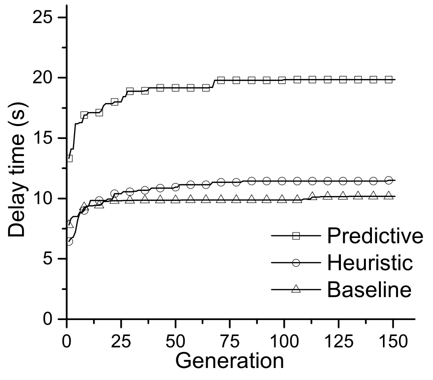
(b) Scenario 2

Figure 5.7: Average *delay time* across 1000 randomly seeded trials for USV teams using *baseline*, *heuristic*, *pred-noswap* or *predictive* strategies.

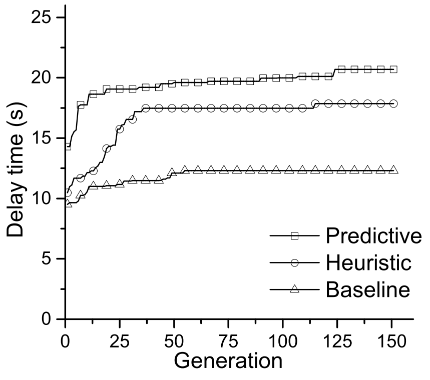
selecting a bad candidate.

To optimize the parameter set Γ for each of the three main strategy types, the genetic algorithm described in Sec. 5.3.5 was performed six separate times, once for every combination of strategy and scenario. An exception was made for the *pred-noswap* strategy, which was given the same parameter set as the *predictive* strategy. The change in performance across 150 generations is shown in Fig. 5.8a and Fig. 5.8b for scenarios 1 and 2, respectively. Most of the gains occurred within the first 50 generations of the algorithm.

Figures 5.9 a) and b) show the average *delay time* as the learning rate δ_{learn} is varied. The utility of the USV team decreases as the learning rate is decreases, and increases as the learning rate increases. This is true for all of the strategies evaluated. However, the *predictive* strategy remains the preferred strategy for all $\delta_{learn} > 0$. The special case where $\delta_{learn} = 0$ means that the USV team is unable to identify the intruders through observation, resulting in a delay time of zero.

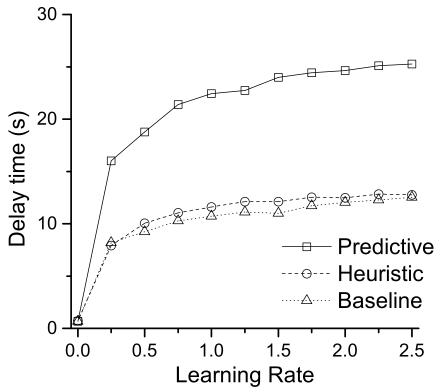


(a) Scenario 1

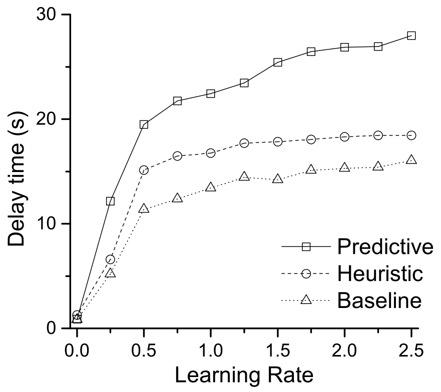


(b) Scenario 2

Figure 5.8: Median *delay time* for each generation of the genetic algorithm when optimizing strategies for scenarios 1 and 2. Results shown for the best-performing chromosome in the population.



(a) Scenario 1

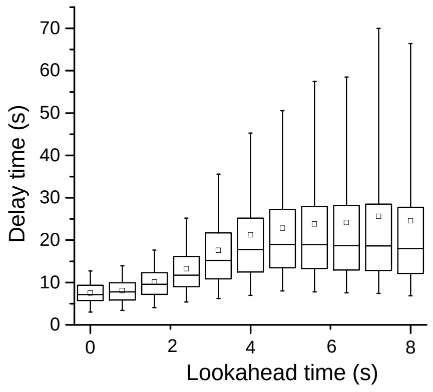


(b) Scenario 2

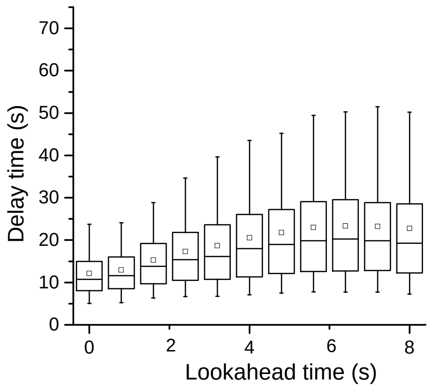
Figure 5.9: Median *delay time* across 1000 randomly seeded trials for *baseline*, *heuristic* and *predictive* strategies as the learning rate δ_{learn} increases.

	Evasive	Non-Evasive
Baseline	9.0 s	21.2
Heuristic	10.1 s	67.6
Predictive	19.0 s	158.3

Figure 5.10: Median *delay time* for *baseline*, *heuristic*, *predictive* strategies in scenario 1 against intruders that perform evasive turns and those that don't.



(a) Scenario 1



(b) Scenario 2

Figure 5.11: Average *delay time* across 1000 randomly seeded trials for the *predictive* strategy as the lookahead time, $t_{lookahead}$, increases.

Fig. 5.10 shows the average *delay time* when each of the strategies is performed against a set of evasive or non-evasive intruders in scenario 1. As expected, non-evasive intruders are delayed from reaching the target for longer than evasive intruders. In practice, this occurred because the non-evasive intruders would become locked in a continuous blocking pattern with a single USV, looping around the target for an extended period of time. The evasive intruder was able to break this pattern by intermittently flipping direction, reaching the target more quickly. We use the evasive intruder model for all the remaining experiments in this chapter.

5.4.3 Running-time Tradeoff

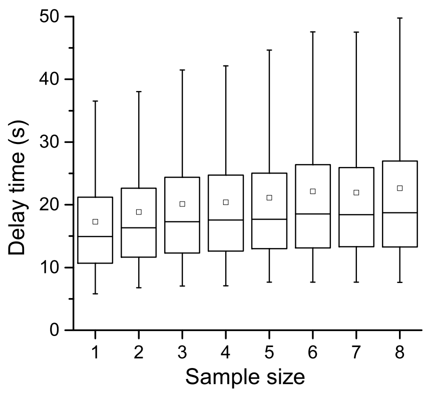
Figures 5.11 and 5.12 show the change in USV team utility as the duration and sample size of the predictive simulation increases. Generally, performing more and better simulations results in higher utility for the USV team, but also results in the algorithm taking longer to execute.

Figures 5.11 a) and b) show the benefit of increasing the lookahead time, $t_{lookahead}$, for the *predictive* strategy in scenario 1 and 2. Longer values $t_{lookahead}$ correspond with more time spent evaluating each predictive simulation. The predictive simulation will run for a maximum duration of $t_{lookahead}$, after which the static evaluation function (defined in Sec. 5.3.4) is performed to quickly estimate the value of the remainder of the simulation. When $t_{lookahead} = 0$, no predictive simulation is run at all, and the static evaluation function is performed immediately.

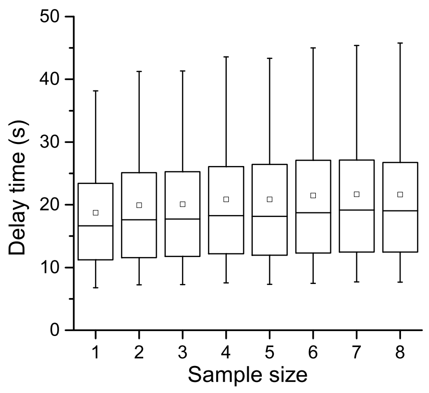
Increasing $t_{lookahead}$ from 0 s to 8 s offers a 151% increase in the median utility for scenario 1, and a 79% increase in the median utility for scenario 2. The benefit of increasing $t_{lookahead}$ starts to diminish at around five seconds, possibly due to the gradual accumulation of errors in the simulation.

Figures 5.12 a) and b) show the benefit of increasing the sample size, n_{sample}, for the *predictive* strategy in scenario 1 and 2. The value of n_{sample} corresponds with the number of possible worlds evaluated via Monte-Carlo sampling. Increasing n_{sample} from 1 to 8 offers a 25% increase in the median utility for scenario 1, and a 14% increase in the median utility for scenario 2.

Increasing $t_{lookahead}$ or n_{sample} individually should result a linear increase in running time for the predictive simulation. However, if the goal is to maximize USV utility, the tradeoff between the quality of the evaluation and the running time of the task re-allocation step should be considered. Both $t_{lookahead}$ and n_{sample} suffer from diminishing returns as their value increases; each additional second added to $t_{lookahead}$ or sample added to n_{sample} is less valuable than the previous. This suggests that the ideal selection of values for $t_{lookahead}$ and n_{sample} will vary depending on the



(a) Scenario 1



(b) Scenario 2

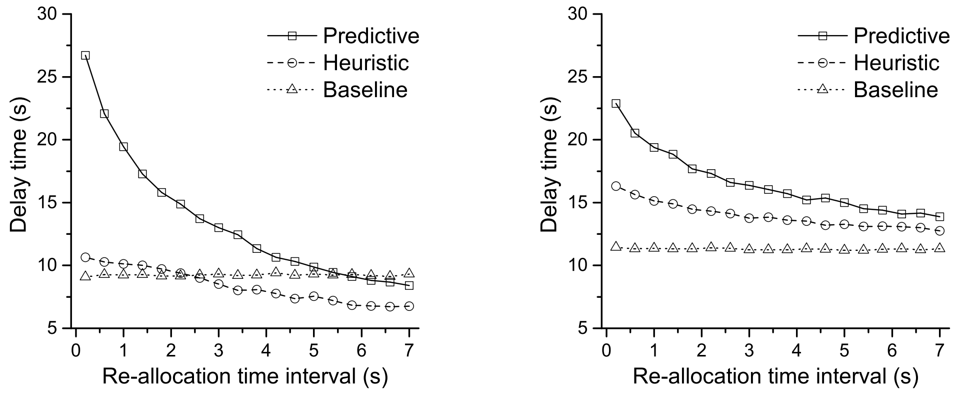
Figure 5.12: Average *delay time* across 1000 randomly seeded trials for the *predictive* strategy as the number of samples n_{sample} increases.

computational power available.

One motivation for minimizing the running time of the predictive simulation is to reduce the time between reallocation steps, t_{alloc}. As shown in Fig. 5.13, reducing t_{alloc} has a positive effect on utility for both the heuristic and predictive strategies. However, Fig. 5.13 also shows that reducing t_{alloc} alone is not sufficient to maximize utility, since the heuristic strategy is out-performed by the predictive strategy even when USVs using the heuristic are allowed to exchange tasks at very high frequency. This suggests that, for sufficiently small values of t_{alloc}, time is better spent carefully evaluating which tasks to exchange instead of exchanging tasks quickly using an inexpensive heuristic.

5.4.4 Scalability

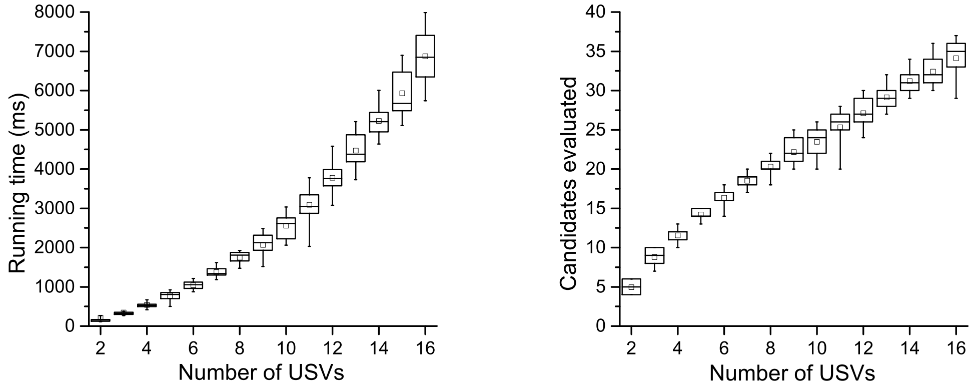
Figs. 5.14 and 5.15 show the effect that increasing the number of USVs or passing boats simultaneously appearing in the scene has on the computational workload



(a) Scenario 1

(b) Scenario 2

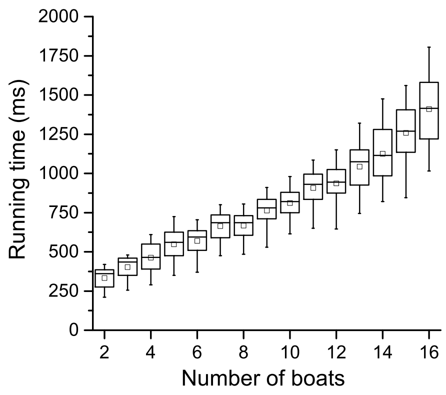
Figure 5.13: Median *delay time* across 1000 randomly seeded trials for the *baseline*, *heuristic* and *predictive* strategies as the re-allocation time interval t_{alloc} increases.



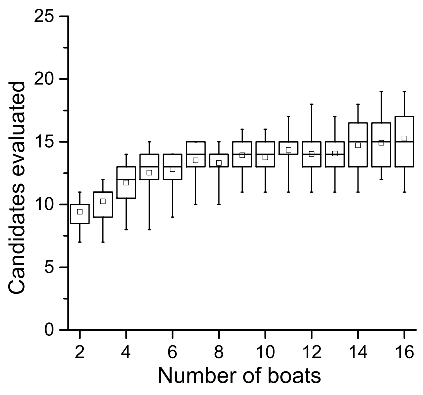
(a)

(b)

Figure 5.14: Average running time and number candidate task allocations evaluated across 1000 randomly seeded trials, as the number of USVs varies, for the *predictive* strategy in scenario 1.



(a)



(b)

Figure 5.15: Average running time and number candidate task allocations evaluated across 1000 randomly seeded trials, as the number of passing boats varies, for the *predictive* strategy in scenario 1.

of *predictive* strategy. Running time was measured using an Intel Core 2 Q6600 Quad processor with a 2.4 GHz clock speed with the algorithm running in a single thread. Results are shown for scenario 1, but similar results should be expected for other scenarios.

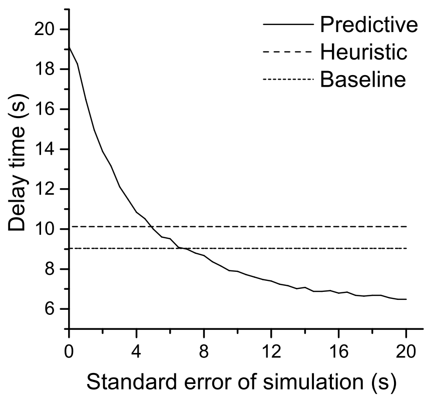
Fig. 5.14 shows that increasing the number of USVs causes a roughly linear increase in the number of candidate task allocations evaluated and a polynomial increase in the running time. As explained in Sec. 5.3.6, the number of candidate task allocations evaluated by predictive simulation is at most $O(|U \times H|)$, where U is the set of USVs and H is the set of tasks. The number of USVs is directly proportional to the number of task exchanges that are considered. Similarly, increasing the number of USVs increases the running time of each predictive simulation. As a result, doubling the number of USVs from 4 to 8 increases running time of the task re-allocation step by 3.4 times, from 525 ms to 1805 ms.

Fig. 5.15 shows the effect of increasing the number of passing boats in scenario 1. Since each passing boat must be assigned an observe or delay task, increasing the number of boats should result in a linear increase in the number of tasks, similar to increasing the number of USVs. However, the effect is much less pronounced than in Fig. 5.14, because a single USV does not have to evaluate every new task that is added. Thus, the cost of adding a USV is greater than the cost of adding a new boat or task to evaluate.

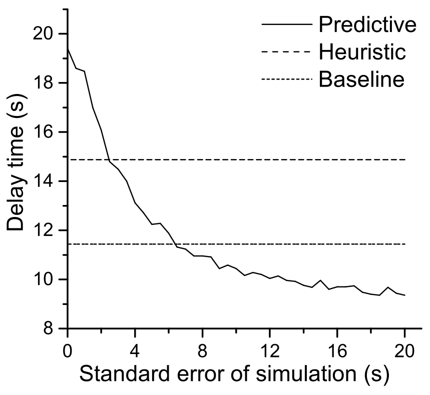
The median running time for a single USV to compute a task re-allocation step when there are 5 USVs and 8 passing boats (the default parameters for scenario 1) was 805 ms, which is within the 1 second allocated in the experimental setup. This result, combined with the low-order polynomial complexity of the algorithm, suggests that the approach is efficient and that online computation is feasible on equivalent hardware.

5.4.5 Robustness

The reliability of the predictive strategy depends on how accurately the predictive simulation is able to estimate the expected value of candidate task allocations. Since the simulator used for our experiments and and the predictive simulator used by USVs both use the same motion model, it is relatively easy for the USVs to estimate the expected value. However, we cannot expect the same level of fidelity in the real world. To determine what effect inaccuracies in the predictive simulation have on performance, we tried several ways to make the simulation less reliable:



(a) Scenario 1



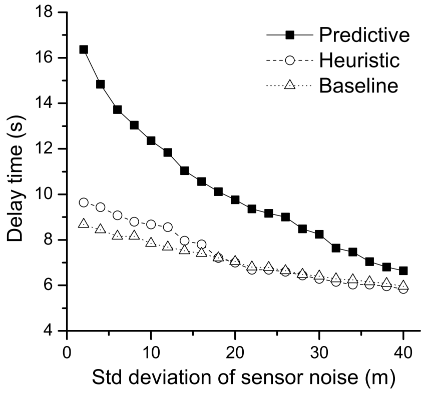
(b) Scenario 2

Figure 5.16: Median *delay time* across 1000 randomly seeded trials for USV teams using the *predictive* strategy when normally distributed error is added to the result of the predictive simulation.

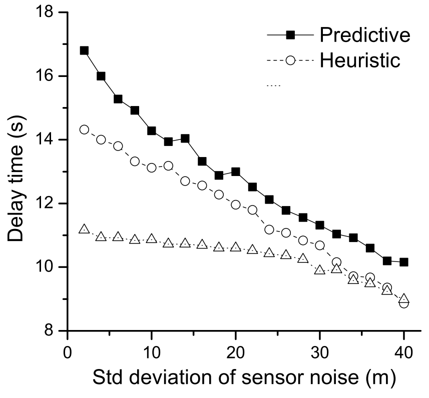
1) adding normally distributed error to the utility value returned by the predictive simulation, 2) adding normally distributed error to the USV sensor measurements and 3) using the incorrect opponent model in the predictive simulation.

The results for the first experiment are shown in Fig. 5.16. There is a significant drop in performance as the utility value returned by the predictive simulation becomes noisier. For scenario 1, the performance of the predictive strategy drops below that of the purely heuristic strategy when standard error exceeds 4.5 seconds. For scenario 2, this occurs when the standard error exceeds 3.0 seconds.

For the results in Fig. 5.17, we added normally distributed error to the USVs' sensor measurements of the locations (x and y coordinates) of the passing boats. The labels on the figure display the standard deviation of this error at an observation range of 80 m or greater. For ranges between 80 m and 0 m, the amount of error was decreased linearly based on distance, so that measurements had no error if taken at 0 m range. As mentioned in Sec. 5.3, each USV uses a Kalman filter to produce



(a) Scenario 1



(b) Scenario 2

Figure 5.17: Median *delay time* across 1000 randomly seeded trials for USV teams using the *predictive* strategy when normally distributed error is added to USV sensor measurements.

estimates of the boats' locations based on this sensor data. The results show a gradual decrease in utility as the sensor measurements become noisier, however the predictive strategy remains the best-performing strategy in spite of the noise.

To evaluate how sensitive the predictive simulation is to the accuracy opponent model, we created two variations on the opponent model by modifying the intruders' level of aggression:

(1.) the "timid" model avoids collisions more actively and performs evasive turns less frequently, increasing γ_{fan_r} and γ_{fan_θ} by 10% and increasing t_{flip} by 100% compared to the normal model,

(2.) the "aggressive" model avoids collisions less actively and performs evasive turns more frequently, decreasing γ_{fan_r} and γ_{fan_θ} by 10% and decreasing t_{flip} by 50% compared to the normal model.

For each opponent model, we produced a complementary predictive strategy,

	Normal	Timid	Aggressive
Pred(N)	19.2 s	17.5 s	13.8 s
Pred(T)	18.5 s	36.9 s	12.8 s
Pred(A)	11.4 s	10.2 s	15.5 s
Pred(R)	18.1 s	28.4 s	13.8 s
Heuristic	10.0 s	10.6 s	9.5 s
Baseline	9.2 s	9.0 s	9.0 s

(a) Scenario 1

	Normal	Timid	Aggressive
Pred(N)	19.3 s	16.9 s	18.2 s
Pred(T)	18.7 s	22.8 s	15.6 s
Pred(A)	16.6 s	14.4 s	20.8 s
Pred(R)	19.0 s	18.2 s	19.0 s
Heuristic	15.0 s	15.4 s	16.1 s
Baseline	11.4 s	11.0 s	13.9 s

(b) Scenario 2

Figure 5.18: Median *delay time* across 1000 randomly seeded trials for USV teams using the *predictive*, *heuristic*, and *baseline* strategies against intruders with timid, normal or aggressive behaviors.

named Pred(T) and Pred(A), where the predictive simulation used the timid, or aggressive model of the intruder respectively. We also produced a fourth strategy, Pred(R), where one of the three intruder models is selected at random at the start of each predictive simulation. For each of the strategies, the GA parameters Γ were tuned for the normal intruder model.

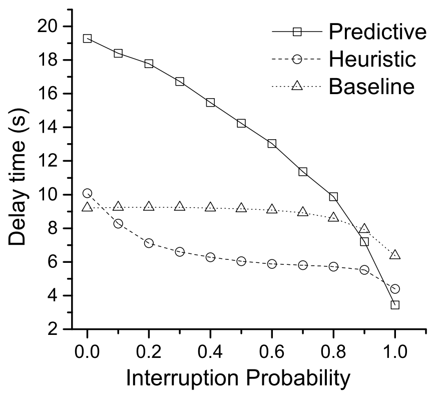
The results in Fig. 5.18 show the median *delay time* when using each of the different task re-allocation strategies against each of the three intruder models. As expected, the USV team performed best when the correct opponent model was used. Additionally, the Pred(R) strategy was the second-best performing strategy in several cases, and performed better than the heuristic strategy in all of the cases.

One key result from this experiment is that using the correct model of the intruder can have a significant impact on the performance of the algorithm. For example, using the Pred(A) strategy against the timid intruder causes the predictive strategy to perform worse than the heuristic strategy. This is because the Pred(A) strategy assumes the intruder is aggressive, when it is actually timid, and makes worse predictions than the other predictive strategies. In contrast, Pred(T) was the best-performing strategy for this situation, while Pred(R) was the second-best.

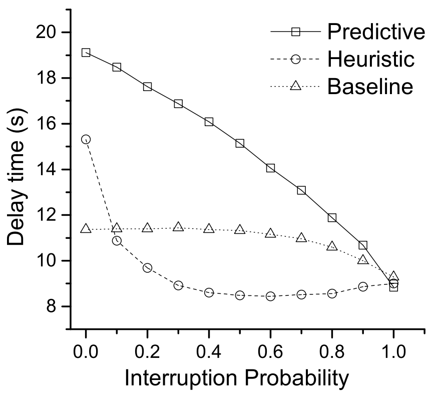
These results suggest that when the intruder model is not exactly known, decent performance can be obtained by sampling over the set of possible intruder models. Additionally, using a model that closely approximates the correct model, (e.g. using the normal model to approximate the timid intruder, or the aggressive model to approximate the normal intruder), results in better utility than using a model that poorly approximates the correct model (e.g. using the aggressive model to approximate the timid intruder). This suggests that the predictive simulator still provides useful information even if the model used does not exactly match the behavior of the opponent.

5.4.6 Interrupted Communication

To determine the effect of communication interruptions on USV team performance, we performed simulation runs where the communication link between each pair of USVs had a random chance of being interrupted. Fig. 5.19 shows the average utility for each of the three strategies when the chance of communication being interrupted is varied. The interruption probability determines the likelihood of an interruption event occurring between a pair of USVs during a 1 s time interval. If an interruption event occurs, the two USVs cannot exchange information for the whole 1 s interval. Each interruption is event modeled as statistically independent,



(a) Scenario 1



(b) Scenario 2

Figure 5.19: Median *delay time* across 1000 randomly seeded trials for USV teams using the *predictive*, *heuristic*, and *baseline* strategies when communication between USVs is interrupted.

so repeated interruptions can block communication between two USVs indefinitely.

As the probability of interruption increases, the performance of the USV team is negatively impacted. This is true for all three strategies. Going from no interruption, to interruptions occurring with 0.5 probability every second, the mean *delay time* of the predictive strategy drops from 19.2 s to 14.2 s in scenario 1, and from 19.1 s to 15.2 s in scenario 2, a difference of 26% and 20% respectively. The impact on the heuristic strategy is more significant, dropping from from 10.1 s to 6.0 s in scenario 1, and from 15.3 s to 8.5 s in scenario 2, a difference of 40% and 44% respectively.

For both scenarios, the difference in performance between the heuristic and predictive strategies decreases as the probability of interruption increases. Since task exchanges are not possible without communication between agents, high interruption renders the additional predictive simulation less effective. The predictive strategy still out-performs both the heuristic baseline strategies in all but the most extreme levels of interruption. The baseline strategy also performs better than the heuristic strategy when the interruption probability exceeds 0.1. This may be due to the fact that the baseline strategy uses very few task exchanges to begin with and is therefore better optimized for situations with low communication.

5.5 Discussion

This chapter introduced a decentralized, contract-based planning approach for protecting an high-valued asset by a team of USVs operating in an environment with civilian traffic. The developed planner is able to deal with uncertainty about which boats are actual intruders and accounts for complex interactions between USVs and intruders when allocating tasks. The planner combines high-level task allocation with low-level user defined behaviors by using model-predictive simulations to evaluate plan performance. The planner is capable of evaluating candidate the task allocation efficiently, is scalable to large teams of USVs, and can be optimized for a specific mission.

We have evaluated the performance of the planner in two different simulation scenarios. In both scenarios, the developed model-predictive planner had a significant performance advantage compared to the baseline and heuristic strategies. We also evaluated the scalability of the planner for large numbers of USVs and passing boats, explored the trade-off between plan quality and execution time, and evaluated the planner's robustness in dealing with noisy sensor data, inaccurate opponent models and high levels of communication interruption.

The experimental results in Sec. 5.4 show that by carefully tuning and integrating the model-predictive simulation into the task allocation process, features such as differential constraints and sensing uncertainty can be directly considered during task allocation and still run efficiently. We also demonstrate that the use of model-predictive simulation leads to significantly higher performance and robustness than the pure use of task-tailored heuristic rules. We show that through careful Monte-Carlo sampling over the distribution of possible worlds, the model-predictive simulation produces better results despite the fact that heuristics are computationally more efficient. These results suggest that time is better spent carefully evaluating which tasks to exchange instead of exchanging tasks quickly based on an inexpensive heuristic.

5.5.1 Limitations and Future Work

One significant limitation of the experimental design in this chapter is that it does not incorporate concurrency into the simulation. The implementation is single-threaded, meaning that the task re-allocation step for each USV is performed sequentially. As a result, some of the concurrency issues that may be experienced in a real world scenario are not directly evaluated by our experiments.

In the presence of concurrency it is possible for the system to cycle between two or more locally sub-optimal task allocations. This can occur when two different USVs perform separate exchanges without being mutually aware of the other's decision. The two exchanges together may have a lower expected utility than the original task allocation, causing the exchanges to be reversed in the subsequent iteration. In this chapter, we did not develop mechanisms specifically to deal with these types of cycles, however, this and other concurrency issues are something that should be evaluated more closely in future work.

Another limitation of the experimental setup is that the simulator used during experiments adheres to the same motion model for boat physics as the predictive simulator used by the USVs to evaluate candidate task allocations. In the real world, it may be unrealistic to expect the same level of fidelity from a predictive simulation. To address this issue in future work, it would be desireable to perform experiments using a higher fidelity simulation environment [64, 65].

In future work, it would be beneficial to explore ways to improve the planner under high communication uncertainty. Since explicit task exchanges between agents are not possible when communication is interrupted, it may be beneficial to blend contract-based task exchanges with purely local task assignment as communication between agents becomes less reliable. Another idea worth investigating is whether the algorithm should always perform the best exchange with the boats currently in communication, or wait a bit longer for openings in communication to perform task exchanges with higher expected value. More work should also be done to evaluate the effect of concurrency on task exchanges in an experimental setting.

Significant gains in performance could be realized by improving how candidate task allocations are sampled during the task re-allocation process. The current algorithm behaves like a local search, evaluating only a small number of changes applied to the current task allocation. Sampling outside this small set of candidates could reduce the likelihood of the task allocation becoming stuck in a local minimum, increasing overall utility for the USV team.

Finally, this approach could be extended to more complex scenarios, such as defending a moving target, accounting for coordinated behavior from the intruders, incorporating static obstacles with complex shapes into the environment, or by applying the algorithm in the ground and aerial vehicles domains. To do this, the blocking and guarding behaviors may need to be modified and new tasks may need to be created to account for changes in the target's position, incorporate path planning to navigate around obstacles, or model the motion constraints of different vehicles. Adding more sophisticated behaviors and corresponding tasks for the current scenarios, such as cooperative blocking for two or more USVs, may also improve performance without any changes to the high level task allocation process.

Chapter 6: Conclusion

This dissertation has presented a variety of novel heuristic planning techniques for multi-agent pursuit-evasion games across several problems domains, including visibility-based pursuit-evasion (Ch. 2&3), target following with differential motion constraints (Ch. 4), and distributed asset guarding using unmanned sea-surface vehicles (Ch. 5). To overcome the inherent difficulties associated with generating solutions for pursuit-evasion problems, these techniques rely heavily on the use of problem relaxation and model predictive simulation to achieve low computational complexity and efficient running times. Experimental results for each heuristic demonstrate favorable performance compared to alternative approaches.

The application of problem relaxation and model predictive simulation to pursuit-evasion games greatly simplifies what can otherwise be a very complicated problem. For many pursuit-evasion games, generating optimal solutions is provably intractable, requiring an exhaustive search over complex information spaces or other computationally demanding problem solving techniques. In contrast, this dissertation demonstrates that the inclusion of a probabilistic opponent model allows for efficient sampling over predicted future states, quickly generating heuristic estimates for use in a planning algorithm.

Although the strategies generated using heuristic approaches do not represent optimal behavior, the experimental results in this dissertation demonstrate practical effectiveness in a number of test cases. Furthermore, the agents encountered in many practical pursuit-evasion games are unlikely to exhibit theoretically optimal behavior. Not only is computing the optimal strategy computationally infeasible, agents are likely to be human, animal, or other subjects whose behavior may be better represented by a probabilistic opponent model that can be learned and integrated into a planning algorithm. The work in this dissertation attempts to demonstrate how that might be achieved for a range of different scenarios.

6.1 Summary of Contributions

Visibility Based Pursuit Evasion

A significant portion of the work presented in this dissertation focuses on visibility-based pursuit-evasion games, with a pair of associated algorithms introduced in Ch. 2 and 3. Specific contributions include:

- A polynomial-time control action selection heuristic for coordinating teams of pursuers in visibility-based pursuit-evasion games (LEL), with an extension to incorporate probabilistic opponent models of the evader (PLA).
- Five different opponent models for the PLA heuristic (M_{RW}, M_{PRW}, M_{EP}, M_{EV}, M_{LP}), including a learned preference opponent model (M_{LP}) which can be trained from interaction traces of previous games.

- An experimental evaluation of the PLA and LEL heuristics versus three different evader strategies (S_{EP}, S_{EV}, S_{RL}), with favorable comparisons against two independent planning approaches (LV, MD).

- A detailed analysis of how varying the parameters of the learned preference opponent model (M_{LP}) affects the PLA heuristic's performance against evaders.

The developed algorithms represent an advancement over prior work due to their low computational complexity and applicability to problem domains that are unsupported by existing research, such as the work by LaValle et al. [2,7]. Although the outlined algorithms are heuristic in nature, they have proven competitive with more rigorous approaches in experimental evaluation, and require comparatively little computational effort. The low computational overhead can be attributed directly to the problem relaxation made in the derivation of the heuristic.

Target Following with Unmanned Boats

A secondary contribution of this dissertation includes a motion goal selection algorithm for pursuit-evasion games between a unmanned sea-surface vehicle (USV) and a target boat. The algorithm performs Monte Carlo sampling over a probabilistic opponent model to generate candidate motion goals for the USV, which are then evaluated using a separate path planning algorithm to ensure dynamically feasible paths. Experimental analysis on randomly generated domains demonstrates that USVs using the motion goal selection algorithm follow more efficient paths than USVs which choose the current state of the target boat as a motion goal

Distributed Asset Guarding

The final major contribution of this dissertation includes a contract-based task allocation algorithm to coordinate a team of USVs defending a high-valued asset from hostile incursion. The team of USVs must identify, intercept and block hostile intruder boats before they reach the asset. Complications of this problem include the presence of differential motion constraints, uncertainty about the identity of intruder boats, and sensor and communication uncertainty.

The task allocation algorithm performs model-predictive simulation to estimate the the value of candidate task allocations, which are then realized by low-level parameterized behaviors. Experimental analysis demonstrates that task assignments selected using model-predictive simulation perform better than task assignments evaluated using pure heuristic rules. The performance also improves as the depth of the prediction and sample size increase. A detailed analysis is provided to illustrate how the performance of this algorithm varies as parameters in the domain and the strategy executed by invaders are subject to change.

Additional Contributions

In addition to the specific contributions listed above, this work illustrates how planning heuristics can be developed at various levels of abstraction, including low-level control action selection (Ch. 2&3), motion goal estimation (Ch. 4), and high-level task allocation (Ch. 5). This work demonstrates how Monte Carlo simulation can be used as a means of facilitating cohesion between independent planning components, such as providing motion goals to path planning algorithms (Ch. 4), or evaluating candidate task allocations through the simulation of low-level parameterized behaviors and control policies (Ch. 5). The exploration of these capabilities may prove useful in the development of similar systems in the future.

Appendix A: Asset Guarding Behavior Model

This section defines the parameterized behaviors and control action selection policies for USVs and boats in Chapter 5.

A.1 Parameterized Behaviors

The behaviors for USV u_i are a set of hand-coded heuristics that map u_i's task assignment H_{u_i} to a unique motion goal $G_{u_i}(O_{u_i}, \mathcal{A}_{u_i})$. The rules for computing this motion goal are defined below.

Each task $h_j \in H_{u_i}$ assigned to USV u_i is given a task-specific motion goal, $G_{h_j}(O_{u_i}, \mathcal{A}_{u_i})$, defined as

$$G_{h_j}(O_{u_i}, \mathcal{A}_{u_i}) = \begin{cases} \text{boat location, } l_{b_j}, & \text{if } h_j \in H_o, \\ \text{guard location, } l_{g_j}, & \text{if } h_j \in H_g, \\ \text{intercept point, } l_{int}, & \text{if } h_j \in H_d. \end{cases} \tag{A.1}$$

For observe or guard tasks, this motion goal corresponds to the estimated boat location l_{b_j} or pre-defined guard location l_{g_j} associated with task h_j. For delay tasks, the motion goal is an intercept point l_{int} positioned along a line between some intruder b_j and the target.

Since USV u_i can be assigned multiple tasks, u_i's desired motion goal may differ from the motion goal of any individual task. The desired motion goal for u_i is

$$G_{u_i}(O_{u_i}, \mathcal{A}_{u_i}) = \begin{cases} G_{h_j}(O_{u_i}, \mathcal{A}_{u_i}), & \text{if } \exists h_j \in H_{u_i} \cap H_d, \\ G_w(O_{u_i}, \mathcal{A}_{u_i}), & \text{otherwise,} \end{cases} \tag{A.2}$$

which returns $G_{h_j}(O_{u_i}, \mathcal{A}_{u_i}) = l_{int}$ if H_{u_i} contains some delay task $h_j \in H_d$. Otherwise, it returns a weighted motion goal (see Fig. A.1a) based on the USV u_i's currently assigned guard and observe tasks,

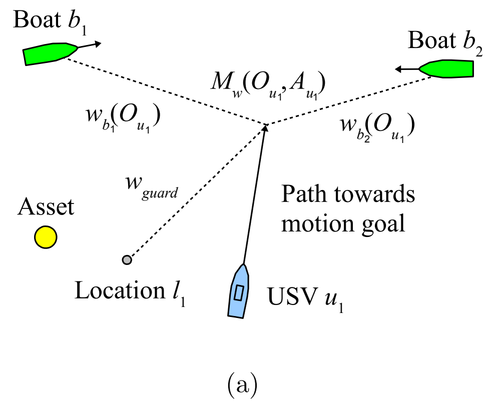
$$G_w(O_{u_i}, \mathcal{A}_{u_i}) = \frac{\sum_{h_j \in H_{u_i}} w_{h_j}(O_{u_i}) G_{h_j}(O_{u_i}, \mathcal{A}_{u_i})}{\sum_{h_j \in H_{u_i}} w_{h_j}(O_{u_i})}, \tag{A.3}$$

where $w_{h_j}(O_{u_i})$ is the weight of task h_j, equal to γ_{guard} if h_j is a guard task, and equal to $w_{b_j}(O_{u_i})$ if h_j is an observe task for boat b_j,

$$w_{b_j}(O_{u_i}) = \gamma_{intr} P(b_j \in I | O_{u_i})(1 + \frac{\gamma_{dist}}{|l_{target} - l_{b_j}|}). \tag{A.4}$$

The parameters γ_{guard}, γ_{intr}, and γ_{dist} are tuned by the genetic algorithm using the method described in Sec. 5.3.5.

We simplify the calculation of intercept point l_{int} by assuming both USVs and intruders can travel in any direction at maximum velocity, ignoring differential constraints and acceleration. If the intruder follows a linear path directly to the target, then l_{int} is the closest intercept point for the USV along that path.



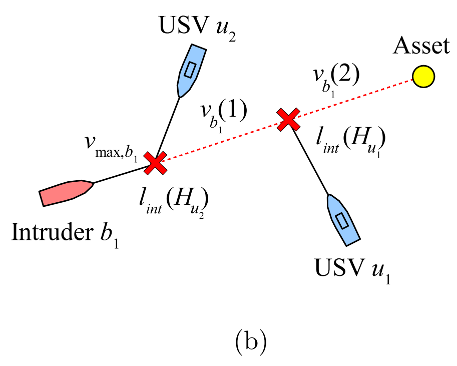


Figure A.1: (a) USV u_1 approaches the weighted motion goal $G_w(\mathcal{O}_{u_i}, \mathcal{A}_{u_i})$ corresponding to two observe tasks for boats b_1 and b_2 and a guard task for location l_1, (b) Heuristic model of USVs u_1 and u_2 intercepting intruder b_1 in a simplified version of the problem. The intercept points serve as a motion goal for the delay behavior of the USVs.

More formally, to find l_{int} for a single USV u_i assigned to a single intruder b_j, the intercept point calculation finds the nearest point in the set of possible intercepts $L_{int}(u_i, b_j)$, defined as

$$L_{int}(u_i, b_j) = \{l : L_{path}(u_i, b_j) \cap L_{target}(b_j)\} \tag{A.5}$$

where $L_{path}(u_i, b_j)$ is the set of points for which some linear path for u_i intercepts some linear path for b_j at their respective maximum velocities,

$$L_{path}(u_i, b_j) = \{l : \frac{|l - l_{u_j}|}{v_{max,u_i}} = \frac{|l - l_{b_j}|}{v_{max,b_j}}\}, \tag{A.6}$$

and L_{target} is the set of points l that lie on the path between b_j and the target,

$$L_{target}(b_j) = \{l : \exists_{s \geq 0}[l_{b_j} + s(l - l_{b_j}) = l_{target}]\}. \tag{A.7}$$

We define $L_{int}(H_{u_i})$ as the union of $L_{int}(u_i, b_j)$ for all boats b_j with a corresponding delay task $h_j \in H_{u_i} \cap H_d$. The point $l_{int}(H_{u_i})$ is the intercept point $l \in L_{int}(H_{u_i})$ that minimizes its distance to u_i's current location.

$$l_{int}(H_{u_i}) = \begin{cases} \arg\min_{l \in L_{int}} |l - l_{u_i}|, & \text{if } L_{int}(H_{u_i}) \neq \emptyset, \\ l_{target}, & \text{otherwise.} \end{cases} \tag{A.8}$$

If $L_{int}(H_{u_i})$ is empty, no intercept is reachable, so USV u_i will head to the location of the target instead.

When multiple USVs are assigned to delay b_j, the calculation of $L_{int}(H_{u_i})$ incorporates a speed reduction for each additional USV that intercepts the intruder. An example of this calculation for multiple USVs is shown in Fig. A.1b. The calculation estimates the speed of the intruder as $v_{b_j}(k) = v_{max,b_j} * (\gamma_{block})^k$ after it has been intercepted by k USVs. The calculation adjusts the intercept points for all subsequent USVs accordingly. Since $v_{b_j}(k)$ underestimates the real travel time of the boats and USVs, we define a lead-time parameter, γ_{lead}, which the calculation adds to USV u_i's starting time when computing the intercept. The parameters γ_{lead} and γ_{block} are tuned by the genetic algorithm using the method described in Sec. 5.3.5.

A.2 Control Action Selection

Given USV u_i's motion goal $G_{u_i}(O_{u_i}, \mathcal{A}_{u_i})$ and vehicle state $\mathbf{x}_{u_i}$, an appropriate control action $q \in Q(\mathbf{x}_{u_i})$ must be selected to direct u_i towards its goal while avoiding collisions with other boats or static obstacles. Let ψ_{u_i} be u_i's current heading angle and let $\phi_{G_{u_i}}$ be the desired heading angle in the direction of G_{u_i}. The steering angle to achieve this new heading is determined by,

$$\Delta\psi_{G_{u_i}}(\mathbf{x}_{u_i}) = \arg \min_{\Delta\psi \in \Theta(\mathbf{x}_{u_i})} |d(\phi_{G_{u_i}}, \psi_{u_i} + \Delta\psi)|. \tag{A.9}$$

where $d(\phi_j, \phi_k)$ is the difference between any two angles ϕ_j and ϕ_k. Similarly, the change in surge speed is determined by,

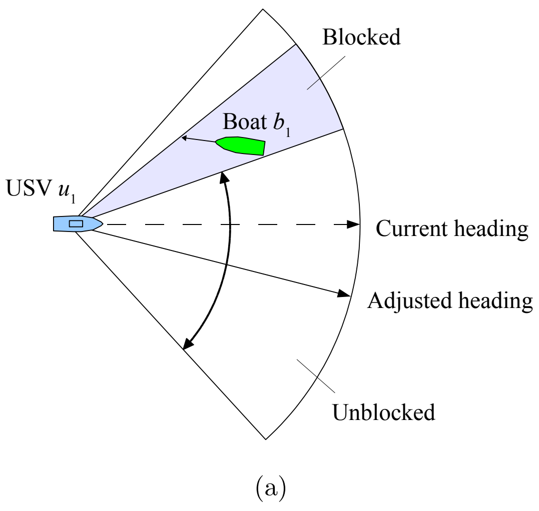
$$\Delta v_{G_{u_i}}(\mathbf{x}_{u_i}) = \arg \min_{\Delta v \in A(\mathbf{x}_{u_i})} |\eta_{G_{u_i}} - (v_{u_i} + \Delta v)| \tag{A.10}$$

where $\eta_{G_{u_i}}$ is the desired surge speed of USV u_i as it approaches G_{u_i}. The resulting control action is simply,

$$q_{G_{u_i}}(\mathbf{x}_{u_i}) = \{\Delta v_{G_{u_i}}(\mathbf{x}_{u_i}), \Delta\psi_{G_{u_i}}(\mathbf{x}_{u_i})\}. \tag{A.11}$$

However, this control action may lead to a collision with obstacles such as other boats or rocks. To reduce the chance of collision, the desired heading $\phi_{G_{u_i}}$ and velocity $v_{G_{u_i}}$ are adjusted using reactive obstacle avoidance.

As depicted in Fig. A.2a, each USV has an obstacle avoidance fan with radius γ_{fan_r} and angular span γ_{fan_θ} to identify which obstacles pose a risk of collision. Headings within the obstacle avoidance fan are considered blocked if they are occupied by an obstacle or will become occupied by an obstacle within some time t_{lead} based on the obstacles' current velocities. Obstacles are assumed to have a non-zero radius.



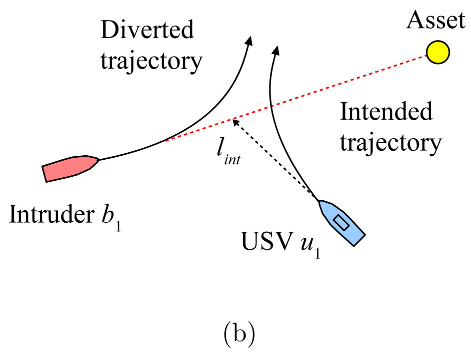


Figure A.2: (a) USV u_1 adjusts its heading to steer away from the region blocked by boat b_1 based on the depicted obstacle avoidance fan, (b) USV u_1 intercepts intruder b_1, diverting it from its intended path to the asset.

Let $Z = \{z_1, z_2, \ldots, z_n\}$ be a set of unblocked sectors inside the obstacle avoidance fan, where each $z_j = [\phi_{j,a}, \phi_{j,b}]$ is a range of headings that are not blocked and where the ordering constraint $\phi_{j,b} \leq \phi_{j+1,a}$ holds for all $j < n$. Let $\phi_{j,mid}$ be a midpoint between $\phi_{j,a}$ and $\phi_{j,b}$. Heading ϕ is considered safe if it is within the obstacle avoidance fan, and $\phi \in [\phi_{1,a}, \phi_{1,mid}]$ or $\phi \in [\phi_{n,mid}, \phi_{n,b}]$, which is trivially true if $n = 1$.

If $\phi^*_{G_{u_i}}$ is the most direct heading to the motion goal, the adjusted heading after reactive obstacle avoidance is,

$$\phi_{G_{u_i}} = \begin{cases} \phi^*_{G_{u_i}}, & \text{if } \phi^*_{G_{u_i}} \text{ is safe,} \\ \phi_{k,mid} \text{ s.t. } z_k = z_{max}, & \text{otherwise,} \end{cases} \tag{A.12}$$

where z_{max} is the widest unblocked sector,

$$z_{max} = \arg\max_{z_j \in Z} |d(\phi_{j,b}, \phi_{j,a})| \tag{A.13}$$

The surge speed of the USV is not affected by obstacle avoidance unless $Z = \emptyset$, at which point the USV will slow to a stop. Thus, the desired surge speed is,

$$\eta_{G_{u_i}} = \begin{cases} 0, & \text{if } Z = \emptyset, \\ v_{max,u_i} \frac{|G_{u_i} - l_{u_i}|}{\gamma_{slow_r}}, & \text{if } |G_{u_i} - l_{u_i}| < \gamma_{slow_r}, \\ v_{max,u_i}, & \text{otherwise,} \end{cases} \tag{A.14}$$

where l_{u_i} is u_i's current location, and γ_{slow_r} determines at what distance the USV should start to slow down.

We define non-zero acceptance radius γ_{goal_r} such that USV u_i is considered at its destination if it is within the distance γ_{goal_r} of its motion goal G_{u_i}. The resulting policy for USV u_i is simply,

$$\pi_{u_i}(O_{u_i}, \mathcal{A}_{u_i}) = \begin{cases} q_{G_{u_i}}(\mathbf{x}_{u_i}), & \text{if } |G_{u_i} - l_{u_i}| > \gamma_{goal_r}, \\ q_{stop}(\mathbf{x}_{u_i}), & \text{otherwise,} \end{cases} \tag{A.15}$$

where $q_{stop}(\mathbf{x}_{u_i}) = \{a_{min}(\mathbf{x}_{u_i}), 0\}$ is a control action that quickly halts the movement of the USV.

The control action selection for passing boats is identical to USVs, only a different motion goal G_{b_j} and different set of parameters γ_{fan_r}, γ_{fan_θ} and γ_{slow_r}

are selected. For our experiments detailed in Section 5.4, the parameters for the passing boats including intruders are predefined, while the parameters for the USVs are learned using a genetic algorithm, described in Section 5.3.5.

If one or more USVs move within the obstacle avoidance fan of an another boat, the boat will be forced to adjust its trajectory to avoid a collision. This is illustrated in Fig. A.2b, where a USV intercepts an intruder, diverting its trajectory away from the target.

Bibliography

[1] A.S. LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM (JACM)*, 40(2):224–245, April 1993.

[2] L.J. Guibas, J.C. Latombe, S.M. Lavalle, D. Lin, and R. Motwani. Visibility-based pursuit-evasion in a polygonal environment. In *International Journal of Computational Geometry and Applications (IJCGA)*, pages 17–30. Springer-Verlag, 1997.

[3] J.H. Reif and S.R. Tate. Continuous alternation: The complexity of pursuit in continuous domains. *Algorithmica*, 10:10–157, 1993.

[4] R. Borie, C. Tovey, and S. Koenig. Algorithms and complexity results for pursuit-evasion problems. In *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, pages 59–66, 2009.

[5] A. Kolling, A. Kleiner, M. Lewis, and K. Sycara. Pursuit-evasion in 2.5d based on team-visibility. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4610–4616, Oct 2010.

[6] N.M. Stiffler and J.M. O'Kane. Visibility-based pursuit-evasion with probabilistic evader models. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[7] S.M. LaValle, D. Lin, L.J. Guibas, J.C. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 737–742, 1997.

[8] B.P. Gerkey, S. Thrun, and G. Gordon. Visibility-based pursuit-evasion with limited field of view. In *International Journal of Robotics Research (IJRR)*, pages 20–27, 2004.

[9] B. Tovar and S.M. LaValle. Visibility-based pursuit-evasion with bounded speed. *International Journal of Robotics Research (IJRR)*, 2008.

[10] B.P. Gerkey, S. Thrun, and G.J. Gordon. Visibility-based pursuit-evasion with limited field of view. *International Journal of Robotics Research (IJRR)*, 2006.

[11] R. Murrieta-Cid, R. Monroy, S. Hutchinson, and J.P. Laumond. A complexity result for the pursuit-evasion game of maintaining visibility of a moving evader. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2657–2664, May 2008.

[12] F.V. Fomin and D.M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theoretical Computer Science*, 399(3):236–245, June 2008.

[13] T.H. Chung, G.A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics. *Autonomous Robots*, 31(4):299–316, 2011.

[14] T.D. Parsons. Pursuit-evasion in a graph. *Theory and Applications of Graphs*, pages 426–441, 1976.

[15] N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, and C.H. Papadimitriou. The complexity of searching a graph. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 376–385, Oct 1981.

[16] M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1 – 12, 1984.

[17] R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2):235 – 239, 1983.

[18] A.S. Goldstein and E.M. Reingold. The complexity of pursuit on a graph. *Theoretical Computer Science*, 143(1):93 – 112, 1995.

[19] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion with local visibility. *SIAM Journal on Discrete Mathematics*, 20(1):26–41, 2006.

[20] F.V. Fomin, P. Fraigniaud, and N. Nisse. Nondeterministic graph searching: From pathwidth to treewidth. In *Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 364–375. Springer Berlin Heidelberg, 2005.

[21] J. Chalopin, V. Chepoi, N. Nisse, and Y. Vaxès. Cop and robber games when the robber can hide and ride. *SIAM Journal on Discrete Mathematics*, 25(1):333–359, 2011.

[22] R.J. Nowakowski. Search and sweep numbers of finite directed acyclic graphs. *Discrete Applied Mathematics*, 41(1):1 – 11, 1993.

[23] J. Barát. Directed path-width and monotonicity in digraph searching. *Graphs and Combinatorics*, 22(2):161–172, 2006.

[24] F.V. Fomin, D. Kratsch, and H. Müller. On the domination search number. *Discrete Applied Mathematics*, 127(3):565 – 580, 2003.

[25] P.D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, May 1993.

[26] F.V. Fomin. Helicopter search problems, bandwidth and pathwidth. *Discrete Applied Mathematics*, 85(1):59 – 70, 1998.

[27] S.M. LaValle, C. Becker, and J. Latombe. Motion strategies for maintaining visibility of a moving target. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1997.

[28] H.H. González-Banos, C.Y. Lee, and J.C. Latombe. Real-time combinatorial tracking of a target moving unpredictably among obstacles. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[29] T. Muppirala, S. Hutchinson, and R. Murrieta-Cid. Optimal motion strategies based on critical events to maintain visibility of a moving target. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3826–3831, 2005.

[30] R. Murrieta, A. Sarmiento, S. Bhattacharya, and SA Hutchinson. Maintaining visibility of a moving target at a fixed distance: The case of observer bounded speed. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 479–484, 2004.

[31] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on Computing*, 21:863–888, 1992.

[32] A. Abdelrazek and H. El-Alfy. Visibility induction for discretized pursuit-evasion games. In *AAAI Conference on Artificial Intelligence*, 2012.

[33] S. Bhattacharya and S. Hutchinson. On the existence of nash equilibrium for a two-player pursuitevasion game with visibility constraints. *International Journal of Robotics Research (IJRR)*, 29(7):831–839, 2010.

[34] S. Sachs, S.M. LaValle, and S. Rajko. Visibility-based pursuit-evasion in an unknown planar environment. *International Journal of Robotics Research (IJRR)*, 2004.

[35] E. Raboin, U. Kuter, D.S. Nau, S.K. Gupta, and P. Švec. Strategy generation in multi-agent imperfect-information pursuit games. *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 947–954, 2010.

[36] E. Raboin, U. Kuter, and D.S. Nau. Generating strategies for multi-agent pursuit-evasion games in partially observable euclidean space. *Autonomous Robots and Multirobot Systems (ARMS)*, 2010.

[37] J.A. Sethian. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Sciences (PNAS)*, pages 1591–1595, 1995.

[38] K. Hormann and A. Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry: Theory and Applications*, 20(3):131–144, November 2001.

[39] L. Yatziv, A. Bartesaghi, and G. Sapiro. O(n) implementation of the fast marching algorithm. *Journal of Computational Physics*, 212(2):393 – 399, 2006.

[40] S.H. Kim, J.H. Park, S.H. Choi, S.Y. Shin, and K.Y. Chwa. An optimal algorithm for finding the edge visibility polygon under limited visibility. *Information Processing Letters*, 53(6):359 – 365, 1995.

[41] J.B. Kruskal Jr. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):pp. 48–50, 1956.

[42] James A. Sethian and A. Mihai Popovici. 3-D traveltime computation using the fast marching method. *Geophysics*, 64(2):516–523, 1999.

[43] L. Janson, E. Schmerling, A. Clark, and M. Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *International Journal of Robotics Research (IJRR)*, 2015.

[44] E. Reinhard and F.W. Jansen. Rendering large scenes using parallel ray tracing. *Parallel Computing*, 23(7):873–885, July 1997.

[45] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F (Radar and Signal Processing)*, volume 140, pages 107–113(6), April 1993.

[46] S. Thrun. Particle filters in robotics. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, 2002.

[47] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 593–598, 2002.

[48] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(12):99 – 141, 2001.

[49] A. Parker, D.S. Nau, and V.S. Subrahmanian. Overconfidence or paranoia? Search in imperfect-information games. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2006.

[50] S. Shimoda, Y. Kuroda, and K. Iagnemma. Potential field navigation of high speed unmanned ground vehicles on uneven terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2005.

[51] B. Massey. Fast perfect weighted resampling. In *Acoustics, Speech and Signal Processing (ICASSP)*, pages 3457–3460. IEEE, 2008.

[52] C.F. Chung and T. Furukawa. Coordinated pursuer control using particle filters for autonomous search-and-capture. *Robotics and Autonomous Systems*, 57(6-7):700–711, June 2009.

[53] P. Švec, M. Schwartz, A. Thakur, and S.K. Gupta. Trajectory planning with look-ahead for unmanned sea surface vehicles to handle environmental disturbances. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, September 2011.

[54] A. Thakur and S.K. Gupta. Real-time dynamics simulation of unmanned sea surface vehicle for virtual environments. *Journal of Computing and Information Science in Engineering*, 11:031005, 2011.

[55] P. Švec, A. Thakur, E. Raboin, C. Shah, B, and S.K. Gupta. Target following with motion prediction for unmanned surface vehicle operating in cluttered environments. *Autonomous Robots*, 36(4):383–405, 2013.

[56] P. Švec, A. Thakur, and S.K. Gupta. USV trajectory planning for time varying motion goal in an environment with obstacles. In *ASME 2012 International Design Engineering Technical Conferences (IDETC) & Computers and Information in Engineering Conference (CIE)*, August 2012.

[57] M. Bibuli, M. Caccia, L. Lapierre, and G. Bruzzone. Guidance of unmanned surface vehicles: Experiments in vehicle following. *Robotics & Automation Magazine, IEEE*, (99):1–1, 2012.

[58] M. Breivik, V.E. Hovstein, and T.I. Fossen. Straight-line target tracking for unmanned surface vehicles. *Modeling, Identification and Control*, 29(4):131–149, 2008.

[59] J. Majohr and T. Buch. Modelling, simulation and control of an autonomous surface marine vehicle for surveying applications measuring dolphin messin. *IEE Control Engineering Series*, 69:329, 2006.

[60] Z. Peng, D. Wang, Z. Chen, X. Hu, and W. Lan. Adaptive dynamic surface control for formations of autonomous surface vehicles with uncertain dynamics. *IEEE Transactions on Control Systems Technology*, (99):1–8, 2012.

[61] M. Breivik, V.E. Hovstein, and T.I. Fossen. Ship formation control: A guided leader-follower approach. In *World Congress*, volume 17, pages 16008–16014, 2008.

[62] P. Švec and S.K. Gupta. Automated synthesis of action selection policies for unmanned vehicles operating in adverse environments. *Autonomous Robots*, 32(2):149–164, 2012.

[63] E. Raboin, P. Švec, D. S. Nau, and S.K. Gupta. Model-predictive target defense by team of unmanned surface vehicles operating in uncertain environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[64] A. Thakur and S.K. Gupta. Real-time dynamics simulation of unmanned sea surface vehicle for virtual environments. *Journal of Computing and Information Science in Engineering*, 11(3):031005, 2011.

[65] A. Thakur, P. Švec, and S.K. Gupta. GPU based generation of state transition models using simulations for unmanned surface vehicle trajectory planning. *Robotics and Autonomous Systems*, 2012.

[66] L.E. Parker. Multiple mobile robot systems. *Springer Handbook of Robotics*, pages 921–941, 2008.

[67] M.B. Dias, R. Zlot, N. Kalra, and A. Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006.

[68] B.P. Gerkey and M.J. Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *International Journal of Robotics Research (IJRR)*, 23(9):939–954, 2004.

[69] A.R. Mosteo and L. Montano. A survey of multi-robot task allocation. *Instituto de Investigación en Ingeniería de Aragón*, 2010.

[70] R. Simmons, D. Apfelbaum, D. Fox, R.P. Goldman, K.Z. Haigh, D.J. Musliner, M. Pelican, and S. Thrun. Coordinated deployment of multiple, heterogeneous robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2254–2260. IEEE, 2000.

[71] Y. Shoham and K. Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2010.

[72] R.G. Smith. The contract net protocol: high-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 100(12):1104–1113, 1980.

[73] T. Sandholm. Contract types for satisficing task allocation. In *Proceedings of the AAAI spring symposium: Satisficing models*, pages 23–25, 1998.

[74] B.P. Gerkey and M.J. Matarić. Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, 2002.

[75] M.B. Dias. *Traderbots: A new paradigm for robust and efficient multirobot coordination in dynamic environments*. PhD thesis, Carnegie Mellon University, 2004.

[76] R. Zlot and A. Stentz. Market-based multirobot coordination for complex tasks. *International Journal of Robotics Research (IJRR)*, 25(1):73–101, 2006.

[77] N. Kalra, D. Ferguson, and A. Stentz. Hoplites: A market-based framework for planned tight coordination in multirobot teams. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1170–1177. IEEE, 2005.

[78] F. Tang and L.E. Parker. A complete methodology for generating multi-robot task solutions using ASyMTRe-D and market-based task allocation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3351–3358, 2007.

[79] F. Fang, A.X. Jiang, and M. Tambe. Designing optimal patrol strategy for protecting moving targets with multiple mobile resources. In *International Workshop on Optimisation in Multi-Agent Systems (OPTMAS)*, 2013.

[80] B. Bošanský, V. Lisý, M. Jakob, and M. Pěchouček. Computing time-dependent policies for patrolling games with mobile targets. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[81] M. Jakob, O. Vaněk, O. Hrstka, and M. Pěchouček. Agents vs. pirates: multi-agent simulation and optimization to fight maritime piracy. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 37–44. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[82] E.A. Shieh, B. An, R. Yang, M. Tambe, C. Baldwin, J. DiRenzo, B. Maule, and G. Meyer. Protect: An application of computational game theory for the security of the ports of the united states. In *AAAI Conference on Artificial Intelligence*, 2012.

[83] O. Vanek, B. Bosansky, M. Jakob, V. Lisy, and M. Pechoucek. Extending security games to defenders with constrained mobility. In *Proceedings of AAAI Spring Symposium GTSSH*, 2012.

[84] Y. Zhang and Y. Meng. A decentralized multi-robot system for intruder detection in security defense. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5563–5568. IEEE, 2010.

[85] E. Simetti, A. Turetta, G. Casalino, E. Storti, and M. Cresta. Protecting assets within a civilian harbour through the use of a team of USVs: Interception of possible menaces. OCEANS, 2010.

[86] D. Portugal and R. Rocha. A survey on multi-robot patrolling algorithms. *Technological Innovation for Sustainability*, pages 139–146, 2011.

[87] D. Portugal and R.P. Rocha. On the performance and scalability of multi-robot patrolling algorithms. In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 50–55. IEEE, 2011.

[88] J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* MIT Press, Cambridge, MA, USA, 1992.