ABSTRACT

Title of Dissertation:	CONTROLLER SYNTHESIS AND FORMAL BEHAVIOR INFERENCE IN AUTONOMOUS SYSTEMS Estefany Carrillo Doctor of Philosophy, 2021
Dissertation Directed by:	Professor Huan Xu Department of Aerospace Engineering

Autonomous systems are widely used in crucial applications such as surveillance, defense, firefighting, and search & rescue operations. Many of these application require systems to satisfy user-defined requirements describing the desired system behavior. Given high-level requirements, we are interested in the design of controllers that guarantee the compliance of these requirements by the system. However, ensuring that these systems satisfy a given set of requirements is challenging for many reasons, one of which is the large computational cost incurred by having to account for all possible system behaviors and environment conditions. These computational difficulties are exacerbated when systems are required to satisfy requirements involving large numbers of tasks emerging from dynamic environments. In addition to computational difficulties, scalability issues also arise when dealing with multi-agent applications, in which agents require coordination and communication to satisfy mission requirements. This dissertation is an effort towards addressing the computational and scalability challenges of designing controllers from high-level requirements by employing reactive synthesis, a formal methods approach, and combining it with other decision-making processes that handle coordination among agents to alleviate the load on reactive synthesis. The proposed framework results in a more scalable solution with lower computational costs while guaranteeing that high-level requirements are met. The practicality of the proposed framework is demonstrated through various types of multi-agent applications including firefighting, fire monitoring, rescue, search & rescue and ship protection scenarios.

Our approach incorporates methodology from computer science and control, including reactive synthesis of discrete systems, metareasoning, reachability analysis and inverse reinforcement learning. This thesis consists of two key parts: reactive synthesis from linear temporal logic specifications and specification inference from demonstrations of formal behavior. First, we introduce the reactive synthesis problem for which the desired system behavior specifies the method by which a multi-agent system solves the problem of decentralized task allocation depending on communication availability conditions. Second, we present the synthesis problem formulated to obtain a high-level mission planner and controller for managing a team of agents fighting a wildfire. Third, we present a framework for inferring linear temporal logic specifications that succinctly convey and explain the observed behavior. The gained knowledge is leveraged to improve motion prediction for agents behaving according to the learned specification. The effectiveness of the inference process and motion prediction framework are demonstrated through a scenario in which humans practice social norms commonly seen in pedestrian settings.

CONTROLLER SYNTHESIS AND FORMAL BEHAVIOR INFERENCE IN AUTONOMOUS SYSTEMS

by

Estefany Carrillo

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
2021

Acknowledgments

I owe my deepest gratitude to my advisor, Huan Xu, for all the support, guidance and encouragement she has provided throughout my PhD studies. She has always encouraged my research endeavors and given me the freedom to pursue my own interests.

It has been a great pleasure and an honor for me to have Jeffrey W. Herrmann, Robert Celi, Robert Sanner and Michael Otte on my thesis committee. A very important portion of this work has been inspired by the discussions with them. I give special thanks to Dr. Otte, who took on the role of my advisor during my advisor's temporary absence. I am also grateful for the opportunity to work with excellent collaborators, Joshua Shaffer, Suyash Yeotikar, Sharan Nayak, Mohamed Khalid M. Jaffar, Eliot Rudnick-Cohen, and Ruchir Patel.

A special thanks goes to Vidya Raju and Lina Castano for their tremendous help and advice throughout my graduate journey. I would also like to thank the AFRL, Amazon Lab 126 and the wonderful group of people in the Department of Aerospace Engineering. Finally, I want to thank God who has shown me unconditional love through my family, especially my parents and my husband, to whom I dedicate this thesis.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

MAS	Multi-agent Systems
LTL	Linear Temporal Logic
MDP	Markov Decision Process
GR(1)	General Reactivity(1)
BDD	Binary Decision Diagram
PA	Probabilistic Automaton
IRL	Inverse Reinforcement Learning
BA	Büchi Automaton

Chapter 1: Introduction

1.1 Motivation

Modern autonomous systems, operating in dynamic environments, are often required to satisfy a user-provided set of high-level requirements describing the desired system behavior. An expressive and powerful specification language of linear temporal logic (LTL) is used throughout this dissertation to specify these requirements. A benefit of using a formal language like LTL is the existence of many methods and tools that provide automatic synthesis and verification of the system behavior for problems involving a wide range of requirements. Reactive synthesis, for instance, is a method for generating controllers of complex systems using formal specifications written in LTL. The primary benefit and motivation for research using this method is the correct-by-construction attribute: the synthesized controllers take into account system and environment variables with varying dynamics and initial conditions, and are guaranteed to meet the specifications for a system, assuming the environment behaves as formally described. Hence, programmers are not required to "handcraft" individual behaviors of a system under specific conditions (of which are often error prone) and can instead focus on defining the system and specifications in relation to an environment. The major difficulty of using reactive synthesis is the computational burden in dealing with large numbers of environment and system variables, especially observable when considering dynamic environments. This primarily stems from the state explosion that occurs when analyzing all possible interactions between the system and environment. In this thesis, we will address some of the challenges that arise when applying reactive synthesis to design control protocols for multi-agent systems. Some of these challenges include computational complexity, scalability, unreliable communication, and presence of obstacles in the environment.

We are also interested in the inverse problem of reactive synthesis, which is, given demonstrations of behavior satisfying some unknown formal specification, our goal is to infer the specification that best conveys the demonstrated behavior. Consider as a motivating example a scenario in which an agent attempts to satisfy a specification stated as, “reach a yellow tile while avoiding red tiles” (see Fig. 1.1). By observing multiple demonstrations of the agent’s behavior, operating in stochastic settings, our goal is to infer the specification that succinctly conveys and explains the agent’s intent. Often, there is no “correct” answer, and the task specification is best described as a belief over multiple LTL specifications. Learning the most likely LTL specifications from the observed behavior can be leveraged to enable artificial agents to learn to perform the same task as the demonstrator without requiring the user to explicitly provide specifications or objectives in advance, which can be nontrivial and time intensive [96]. Furthermore, if the inferred behavior is expressed as an LTL specification, this specification can be used to automatically synthesize a policy for agents to execute. In addition, inference results can be leveraged to improve motion prediction for agents performing the learned task. This work primarily focuses on leveraging inference results for improving motion prediction of humans. The practicality of the proposed inference framework is demonstrated through a scenario in which humans practice social norms commonly seen in pedestrian settings. This thesis proposes that by inferring social norms that govern human's motion behavior, we can gain a clearer understanding of the intent of the human and account for the learned knowledge in the computation of the belief over the human's future states.

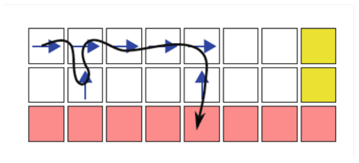


Figure 1.1: Example of an agent demonstrating its intent to satisfy the specification "reach a yellow tile while avoiding red tiles" (adapted from [100]).

1.2 Related Work

Formal methods have been extensively studied in both computer science and control. These approaches rely on applying mathematically-based techniques in proving system correctness [65]. A challenge in the formal methods domain lies in the combinatorial blow up of the state space, commonly known as the state explosion problem. Recently, the development of a polynomial-time algorithm to construct finite state automata from temporal logic specifications enables automatic synthesis of discrete systems that satisfy a large class of properties including safety, guarantee and response even in the presence of an adversary (typically arising from changes in the environments) [82]. Literature refers to this polynomial-time algorithm as reactive synthesis. Applying reactive synthesis requires obtaining abstractions of dynamical systems usually done via abstraction-based hierarchical approaches [60], [61], [64]. For problems considering a large number of system and environment states, a high-level controller created with just reactive synthesis could quickly present an impractical solution. Indeed, research has largely avoided using reactive synthesis for problems with a large number of environmental variables. When the high-level design of a system involves large scale environmental permutations and state spaces, solutions typically seek to discretize the synthesis problem or approach the problem from a different perspective. Discretization appears in the use of receding horizon control in [63] and the use of decentralized controllers for multiple agents in [15]. Both examples break down the top-level synthesis problem into smaller, discrete pieces for the computation benefits. On the other hand, [16] approached their synthesis problem with a focus on resolving deadlock under specific environment conditions instead of directly avoiding dynamic obstacles. In each of the presented cases, the problem description focused on a limited task space (i.e. the number of progress goals) and how the solution can handle larger sets of actions from an environment in relationship to safety specifications.

Further expanding this concept to a whole team of UAVs, the problem worsens due to an increase in the number of system variables proportional to or greater than the number of UAVs, if considering centralized controllers. Even with decentralized controllers for each UAV, additional system variables might need to be introduced to describe coordination and behaviors between the decentralized UAV controllers.

To alleviate the computational complexity on reactive synthesis in this regard, the coordination of UAVs can be handled by an alternative process. In this thesis, we have proposed dynamic allocation as the alternative process for a firefighting application and decentralized task allocation for surveillance and search and rescue applications. Thus, we tackle different synthesis problem formulations with varying levels of computational complexity by applying reactive synthesis within a metareasoning framework. In this framework, each agent in the team adopts a multi-layer reasoning model consisting of *metareasoning*, *reasoning* and *doing*. At the reasoning layer, we apply reactive synthesis to generate a task and path planner for each agent. At the metareasoning layer, we employ reactive synthesis to generate a multi-agent metareasoning policy.

Metareasoning research encompasses various approaches to reason about one's own thinking, memory and processing in order to control different aspects of reasoning such as strategy selection and allocation of resources. In the context of cooperative MAS, metareasoning approaches have been applied to coordinating the agents' behavior, bringing new challenges as a result of agents performing additional reasoning from which benefits gained might depend on the reasoning and behaviors of other agents [49]. For instance, Raja and Lesser [38] framed multi-agent metareasoning as a decentralized coordination problem in which agents maintain a model of each other's meta-level control and coordinate the use of their reasoning resources. George et al. [52] investigated an organizational design approach to coordinate both the agents' behavior and their reasoning by identifying high-performing behavioral patterns and prohibiting agents from reasoning about behaviors counter to these patterns. The effectiveness of metareasoning was shown in [39] for coordinating a team of agents in a tornado tracking application, while [33] applied a centralized controller to coordinate agents with limited communication among them. In contrast to these metareasoning approaches which try to dynamically assess the benefit of additional reasoning at the cost of computational complexity, the metareasoning policy proposed in this thesis is trained completely offline and synthesized from LTL specifications, limiting the computational effort required for online execution of the policy.

1.3 Organization of the thesis and Summary of Contributions

The work in this dissertation is focused on synthesizing controllers from user-defined system requirements. The concept of system here refers to autonomous systems (e.g. UAVs, humans). A simple motivating example is a firefighting scenario in which a team of UAVs is tasked with monitoring and dropping suppressant on designated fire locations, returning to base for refilling suppressant, and emergency landing when a system failure is detected. For such scenario, we are interested in synthesizing a high-level planner that a team of UAVs can execute to satisfy the given requirements within a formal methods framework. The key benefit of such framework is the formal verification of the system behavior, governed by the synthesized controller, with respect to the given requirements.

In this thesis, we consider single and multi-agent systems and their application to modern day operations such as surveillance, fire monitoring, search and rescue and ship protection operations. For Part I of this thesis, we explore multi-agent systems and adopt for each agent a multi-layered model with three different layers of reasoning. This enables us to formulate the problem of controller synthesis at each layer of reasoning and demonstrate the benefits and challenges of applying a formal methods approach to solve for a controller with formal guarantees. For Part II of this thesis, we consider single-agent systems in which the agent is assumed to be a human. Here, we are interested in analyzing the human's motion behavior satisfying an LTL specification in pedestrian settings. Our goal is to infer the specification that best explains the observed pedestrian behavior.

Thus, the research presented here consists of two key components *controller synthesis* applied at each layer of reasoning and *specification inference* from demonstrations of formal behavior. Specification refers to a precise description of both the system and its desired properties. To simplify the analysis of the system, we apply hierarchical based abstractions techniques to capture only the relevant aspects of the system [53].

This thesis has two main parts. The first part focuses on the controller synthesis aspect while the second part focuses on the specification inference aspect. Specification is mentioned in both parts as a key requirement that enables systematic verification and design. The original contributions of this work cover both theoretical and application aspects as outlined below.

Chapters 2 summarizes relevant concepts on metareasoning, linear temporal logic and reactive synthesis. A brief overview of LTL notation is also provided along with the type of specifications that are considered throughout this work. This chapter ends with the general formulation of the reactive synthesis problem along with the system and environment definitions.

Part I: From LTL Specifications to Formal Behavior

Chapter 3 contains the main contributions of the part of the thesis describing the controller synthesis problem formulated at the meta-level layer of each agent in a MAS. Here, the goal of controller synthesis is to generate a metareasoning policy (meta-level control) from LTL specifications encoding the team's prescribed behavior for task allocation as a function of the observed communication quality in the environment. By applying reactive synthesis to generate the meta-level control, we obtain a policy that is correct-by-construction with respect to these LTL specifications. Specifically, the contribution of this part of the thesis with respect to applying reactive synthesis is the appropriate finite state abstraction for the system and environment such that the complexity of the problem is kept low. The organization of this chapter is as follows. First, it describes the decentralized task allocation algorithms that each agent can execute to coordinate tasks with other agents in the system. This description is followed by a discussion of the communication model used to simulate varying levels of communication as well as a detailed description of each type of task allocation scenario considered for evaluation of the policy. Then, we present the metareasoning framework within which the reactive synthesis problem is formulated and solved. For each scenario, we demonstrate that the proposed multi-agent metareasoning approach achieves enhanced performance (i.e. lower communication and travel costs) when solving the problem of decentralized task allocation compared to running a single task allocation algorithm. The material presented in this part of the thesis has been submitted and published in [6] and [32], respectively.

The main contribution of Chapter 4 is in the application of reactive synthesis at the object-level layer of each agent in a MAS. Here the goal of controller synthesis is to generate a high-level mission planner and controller for managing unmanned aerial vehicles (UAVs) fighting a wildfire through the hierarchical integration of reactive synthesis, used for assuring desired system design traits, and dynamic allocation, used for making heuristic-based decisions. This hierarchical approach makes the synthesis problem more tractable and its solution more scalable. Reactive synthesis provides a formal means of guaranteeing the correctness of the UAVs' choice of object-level actions corresponding to its transitions to areas of fire, refill of water, and land as defined by the linear temporal logic specifications. Dynamic allocation coordinates the behavior of multiple UAVs through assignments to regions of fire based on a cost function that takes into consideration the fire locations relative to a UAV, distance to the domain edge, wind speed and direction, and the amount of suppressant already present. The use of receding horizons in the reactive synthesis formulation helps address the computational complexity challenges. Modifications to these horizon definitions guarantee that the scenario still maintains the overall realizability of the formal specifications after the inclusion of static obstacles. Through various scenarios, we demonstrate the effectiveness of multiple UAV fleets in slowing down the progression of fires from reaching the domain edge. The material presented in this part of the thesis has been published in [7].

An additional contribution of Chapter 4 is the implementation of a low-level controller using Hamilton-Jacobi (HJ) reachability analysis. Hamilton-Jacobi (HJ) reachability analysis is a verification method for guaranteeing performance and safety properties of systems [25]. The focus of this dissertation is to to provide safety guarantees and assurances for the UAV executing the discrete transitions dictated by its object-level layer assuming continuous dynamics with disturbance assumptions.

Part II: Specification Inference From Formal Behavior

Chapter 5 shifts focus towards specification inference from demonstrations of formal behavior. Specifically, we evaluate and compare three state of the art inference approaches within an Inverse Reinforcement Learning framework to identify the best performing approach. The inference methods are compared in terms of overall run-time including training time, output type, and main drawbacks. Once the best performing method is identified, extensions are proposed to automatize training data generation and improve the inference process in an iterative manner. The first contribution is the systematic construction process of an initial hypothesis space of specifications. The second contribution is a process of refinement applied to systematically perturb likely specifications to find more complete specifications that capture the demonstrated behavior. The third and last contribution is a human motion prediction framework that uses the inference results to set priors for all possible known goals that the human is expected to visit. Through various scenarios that demonstrate behavior satisfying different LTL specifications, we show the effectiveness of the inference process in obtaining the correct LTL specification and enhanced human motion prediction results in terms of a larger posterior probability for the human's intended goal.

Chapter 2: Background

This chapter provides background for later chapters. This includes a description of the relevant concepts of metareasoning and an overview of linear temporal logic and reactive synthesis.

2.1 Metareasoning

Metareasoning research encompasses various approaches to reason about one's own thinking and decision making process. The goal of metareasoning in monitoring and controlling an agent's reasoning is to improve system performance while reducing computational effort. Extending this concept to multi-agent systems (MAS) has led to *multi-agent metareasoning*, a process whereby a team of agents collectively reasons about the team's decision making process. In MAS, the goal of metareasoning is to improve performance of the MAS instead of the individual agent's performance.

The general metareasoning framework is outlined in [51], which presents a multi-layered agent model (see Fig. 2.1). Within this framework, three types of actions can be considered: *meta-level* actions, *object-level* actions and *ground-level* actions, which we will refer to as *low-level* actions (to avoid confusion with the use of the term "ground-level" in UAV applications) . These actions are defined as follows:

1. *Low-level actions*: These actions are performed by each agent to change its state in the environment. Examples of such actions are movement, communication and sensing.

2. *Object-level actions*: These actions correspond to computational processes that output the low-level action to be performed by the agent to achieve its goal.

3. *Meta-level actions*: These actions are used to analyze and improve the performance of object-level actions.

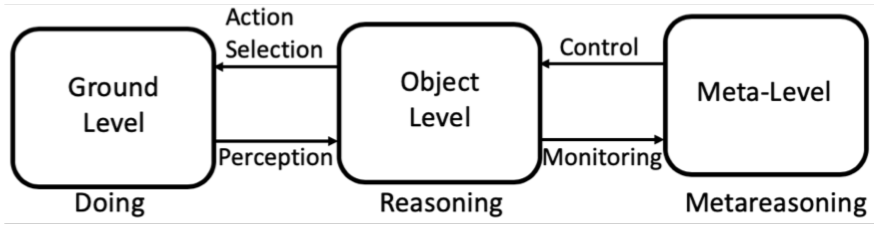


Figure 2.1: Multi-layered agent model [51].

In general, metareasoning approaches for MAS can be categorized by the type of metareasoning structure, problem, and mode they consider [92]. The metareasoning structure describes the way in which metareasoning is implemented to reason about a problem through a metareasoning mode. The mode describes the output from the metareasoning process. In this thesis, we consider the problem of multi-agent coordination. We propose the use of metareasoning to reason about the appropriate algorithm the agents should use when communication availability changes. Our metareasoning approach adopts an independent metareasoning structure, in which each agent in the team has its own meta-level control, which performs metareasoning independently from other agents. Fig. 2.2 shows the independent metareasoning structure with independent meta-levels that do not communicate or coordinate with each other. Instead, communication and coordination happen at the object level.

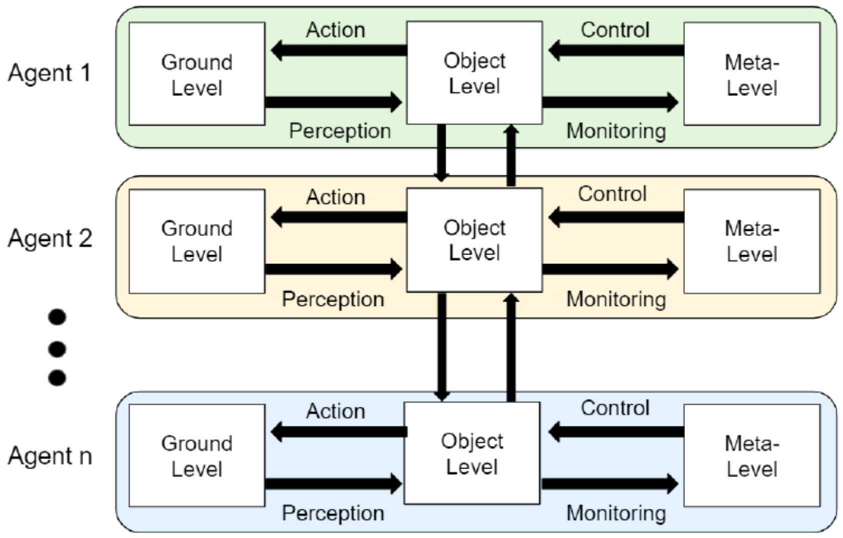


Figure 2.2: Multiagent independent metareasoning structure with n agents that communicate at the object level [92].

Furthermore, at each layer of reasoning, this thesis designs a correct-by-construction controller synthesized from high level requirements written in Linear Temporal Logic (LTL). LTL provides a precise mathematical language for describing the desired requirements.

2.2 Linear Temporal Logic

Linear temporal logic (LTL) is utilized for describing specifications within the formal methods framework. In this thesis, we use specifications involving temporal logic, which is an extension of Boolean logic with temporal semantics. There exist multiple variations of temporal logic languages. LTL is one of the most commonly used languages in the context of robot behavior synthesis. We choose this language because of its expressiveness and the existence of efficient synthesis methods that use LTL.

The building blocks of LTL formulas consist of Boolean variables, logical connectives and temporal modal operators. The logical connectives include: negation ($\neg$), disjunction ($\vee$), conjunction ($\wedge$) and material implication ($\rightarrow$). The temporal modal operators include next ($\bigcirc$), always ($\square$), eventually ($\diamond$) and until ($\mathcal{U}$). By combining these operators, it is possible to specify a wide range of requirements. An *atomic proposition* is a statement on system variables $\mathcal{S}$ that has a unique truth value for a given valuation of the system variables. Let $s \in dom(\mathcal{S})$ be a state of the system and p be an atomic proposition. Then $s \models p$ if p is $True$ at state s. Given a set AP of atomic propositions, LTL formulas are formed according to the following grammar:

$$\varphi := True|p|\neg\varphi|\varphi_1 \ \wedge \ \varphi_2| \bigcirc \varphi|\varphi_1 \, \mathcal{U} \, \varphi_2, \tag{2.1}$$

where $p \in$ AP. Formulas involving *eventually* and *always*, can be derived from the operators shown in Eq. 2.1. LTL formulas over AP are interpreted over infinite sequence of states. Such sequence represents a behavior of the system.

Given a set of atomic propositions AP, we can form the alphabet $\mathbf{\Sigma}_a := 2^{AP}$. We refer to $w = w(0), w(1), w(2), ... \in \mathbf{\Sigma}_a^{\mathbb{N}}$ as an infinite word, a string composed of letters from $\mathbf{\Sigma}_a$. We denote a finite word as w_f. Let L be a labeling function $L : \mathcal{S} \rightarrow \mathbf{\Sigma}_a$ and let the mapped state trajectories $s(k)_{k \geq 0} \in \mathcal{S}^{\mathbb{N}}$ to the set of infinite words $\mathbf{\Sigma}_a^{\mathbb{N}}$ be defined as $w = L(\{s(k)\}_{k\geq 0}) := \{w \in \mathbf{\Sigma}_a^{\mathbb{N}} | w(k) = L(s(k))\}$. Then, given an LTL formula φ, we state the satisfaction relation between w and φ as $w \models \varphi$. We say that φ holds at position $i \geq 0$ of w, if and only if φ holds for the remainder of the sequence starting at position i. Furthermore, $w_k \models \varphi_1 \wedge \varphi_2$ if $w_k \models \varphi_1$ and $w_k \models \varphi_2$. The next operator $w_k \models \bigcirc\varphi$ holds if the property holds at the next time instance φ_{k+1}. The until operator $w_k \models \varphi_1 \,\mathcal{U}\, \varphi_2$ holds if $\exists i \in \mathbb{N} : w_{k+i} \models \varphi_2$, and $\forall j \in \mathbb{N} : 0 \leq j < i, w_{k+j} \models \varphi_1$.

Given propositional formulas p and q, important and widely used properties can be defined in terms of their corresponding LTL formulas as follows,

1. *Safety*: A safety formula is of the form $\Box p$, which asserts that the property p remains true throughout an execution. Typically, a safety property ensures that nothing bad happens. A typical example of safety property frequently used in the robot motion planning domain is obstacle avoidance.

2. *Guarantee*: A guarantee formula is of the form $\Diamond p$, which guarantees that the property p becomes true at least once in an execution. Reaching a goal state is an example of this property.

3. *Progress*: A progress formula is of the form $\Box\Diamond p$, which states that the property p holds infinitely often in an execution. A progress property typically ensures that the system makes progress throughout an execution.

4. *Response*: A response formula is of the form $\Box(p \rightarrow \Diamond q)$, which states that following any point in an execution where the property p is true, there exists a point where the property q is true. For example, a response property can

be used to state how the system should react to changes in the operating conditions.

In many applications, systems need to interact with their environments and whether they satisfy the desired properties depends on the behavior of the environments. In the next section, we informally describe the work of Piterman, et al. [82]. We refer the reader to [82] for the detailed discussion of automatic synthesis of a finite state automaton from system and environment specifications.

2.3 Reactive Synthesis

Reactive systems are systems that maintain an ongoing relation with their environment by appropriately reacting to it. The controllers that regulate the behavior of such systems are called reactive controllers. A control system is a composition of a physical plant, including sensors and actuators, and an embedded controller that runs a control protocol to restrict the behaviors of the plant so that all the remaining behaviors satisfy a set of system specifications.

The synthesis of reactive controls can be interpreted in assume-guarantee form. Given the system specifications, the goal of control synthesis is to find a control logic that, when implemented, ensures that the system satisfies the specifications; or declares that no such logic exists [83].

Let $\mathcal{E}$ and $\mathcal{D}$ be sets of environment and controlled variables, respectively. Let $s = (e, d) \in dom(\mathcal{E}) \times dom(\mathcal{D})$ be a state of the system. Consider an LTL specification of assume-guarantee form, shown in Eq. 2.2,

$$\varphi_e \rightarrow \varphi_s, \tag{2.2}$$

where φ_e characterizes the assumptions on the environment and φ_s characterizes the system requirements. The synthesis problem is then concerned with constructing a control protocol which chooses the move of the controlled variables based on the state sequence so far and the behavior of the environment so that the system satisfies φ_s as long as the environment satisfies φ_e. Let the control protocol be denoted as a partial function $f : (s_0 s_1 ... s_{1-t}, e_t) \rightarrow d_t$. If such a protocol exists, the specification φ is said to be *realizable*. The synthesis problem can be viewed as a two-player game between an environment that attempts to falsify the specification and the system that tries to satisfy it.

For general LTL, the synthesis problem has a doubly exponential complexity [82]. However, a subset of LTL, namely generalized reactivity (1) (GR(1)), is used to solve the synthesis problem in polynomial time (i.e. polynomial in the number of valuations of the variables in $\mathcal{E}$ and $\mathcal{D}$) [83]. GR(1) specifications restrict φ_e and φ_s to have the following form,

$$\begin{aligned} \varphi_e &= \varphi_i^e \ \wedge \ \varphi_t^e \ \wedge \ \varphi_g^e, \\ \varphi_s &= \varphi_i^s \ \wedge \ \varphi_t^s \ \wedge \ \varphi_g^s, \end{aligned} \tag{2.3}$$

where φ_i^e, φ_i^s are the propositional formulas characterizing the initial values for the environment and system variables, φ_t^e, φ_t^s represent transition relations characterizing safe, allowable moves of the state of the environment and system, and φ_g^e, φ_g^s are propositional formulas characterizing goal assumptions for the environment and desired goal specifications for the system that should be attained infinitely often. Given a GR(1) specification, there are game solvers and digital design synthesis tools that output a finite-state automaton that represents the control protocol for the system [81], [23].

This chapter covered the background knowledge of metareasoning, linear temporal logic and reactive synthesis. The following chapters will now dive into the work on designing reactive controllers with formal guarantees of satisfying user-defined requirements under changing environment conditions.

Part I

From LTL Specifications to Formal Behavior

Chapter 3: Meta-Level Layer

In this chapter, we describe the process of formulating reactive synthesis at the meta-level layer of the agent. Our goal is to obtain a multi-agent metareasoning policy that the team executes to decide on how to solve the problem of decentralized task allocation collaboratively. Collaboration in MAS usually requires communication between agents. However, communication is unreliable in realistic environments and outside the control of the multi-robot team, making the coordination of tasks more challenging. Otte *et al.* [31] and Nayak *et al.* [32] have shown that different distributed task allocation algorithms perform better, relative to each other, at different communication quality levels.

To attain robustness to changes in communication, a naive strategy is to have each robot in the team use the best performing algorithm for the perceived level of communication. However, this may cause composability problems. For example, if communication quality varies over the environment, then different team members may select incompatible task allocation algorithms. If communication levels change over time, then switching between different task allocations algorithms may introduce additional overhead, create inefficiencies, or require restarting the task allocation process entirely. This raises questions on whether there are benefits to be gained from switching and which benefits are those.

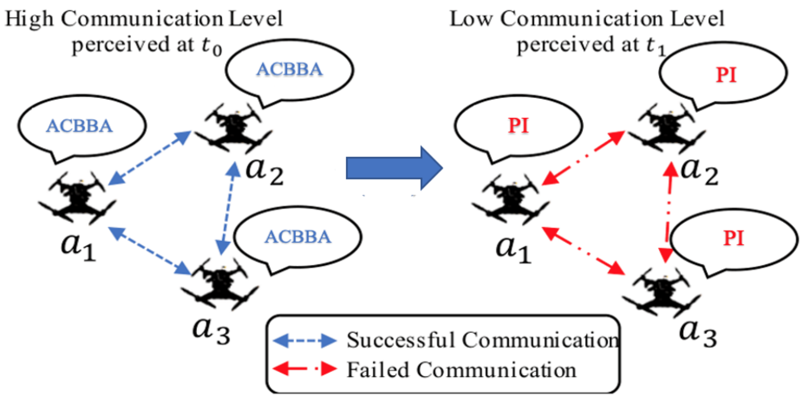


Figure 3.1: A sample switching prescribed by the metareasoning policy for high and low communication levels. On the left, agents a_1, a_2 and a_3 perceive high communication availability at time t_0 and execute ACBBA as their task allocation scheme. On the right, agents switch to performing PI at time t_1 as a result of perceiving low communication availability.

The main motivation of this chapter is to present a meta-level control within a metareasoning framework that addresses these challenges and provides answers to these questions. We propose a multi-agent metareasoning approach that enables a multi-agent team to select which task allocation algorithm to use as a function of changing communication quality level. Given a set of multi-agent task allocation algorithms, we synthesize a policy that prescribes the best algorithm to use among a predefined set of algorithms for a given communication level. We apply reactive synthesis at the meta-level layer to generate the policy from high-level specifications written in Linear Temporal Logic encoding the agents' switching behavior with respect to the state of the environment. Since each agent in the team runs the same policy, the team (or a part of the team) will collectively switch between task allocation algorithms as a function of the observed level of communication. The proposed distributed multi-agent metareasoning policy (Fig. 3.1) is synthesized offline *a priori*, and the resulting policy is executed continually by each agent in real-time, adapting the team's distributed task assignment scheme in response to perceived changes in communication availability.

The contribution of this chapter is a meta-level controller that runs in parallel across a team of agents, to improve system performance under changing communication conditions, when solving the problem of decentralized task allocation. Although meta-level control for the purpose of coordinating agents' behavior has appeared in [38], [39] and [40], here we consider reasoning about a set of existing task allocation methods in order to make the system more robust to varying communication. This additional reasoning can lead to improved task allocation assignments even when communication between the agents deteriorates.

We build the multi-agent metareasoning policy from a set of decentralized task allocation methods for which performance profiles under varying levels of communication were obtained in [32]. This set includes: the Consensus Based Auction Algorithm (**CBAA**) [41], the Asynchronous Consensus Based Bundle Algorithm (**ACBBA**) [42][43], the Decentralized Hungarian Based Algorithm (**DHBA**) [44], the Hybrid Information and Plan Consensus (**HIPC**) algorithm [45][46] and the Performance Impact (**PI**) algorithm [47].

To test the proposed metareasoning policy, we model instantaneous communication conditions using a Rayleigh Fading model [48]. Changes in signal attenuation (i.e., communication quality) over time are modeled by varying the path loss exponent parameter. In each scenario, we simulate high or low communication for some period of time after which, the communication level is switched to low or high, respectively. We consider a full-mesh topology in which every agent *attempts* to communicate with every other agent and then messages are dropped according to the communication model. This assumption is made to facilitate the discovery of performance differences due to applying the proposed policy as opposed to having agents improve communication by changing their network topology as described in previous works [33], [37], [34]. For communication estimation, we propose a straightforward method in which each agent computes an estimate per communication link over which heartbeat messages are expected to arrive from every other agent in the environment. This method is well suited to provide link quality estimates for any underlying communication model. Other sources have explored probabilistic estimation methods of link quality with respect to channel power at different locations, such as in [33], but such estimation is beyond the scope of this paper.

In the following section, we present an overview of the decentralized task allocation algorithms considered for this work.

3.1 Decentralized Task Allocation

Research on decentralized task allocation has been motivated by the vulnerabilities of centralized approaches to communication disruptions and concerns regarding their scalability. Most of the existing decentralized task allocation approaches are consensus-based auction methods, in which agents place bids on tasks and each agent acts as auctioneer and bidder. A comprehensive description of the most commonly used consensus-based decentralized task allocation methods can be found in [41]. Some of these approaches include CBAA and CBBA as well as its asynchronous version, ACBBA [42][43]. Shown to have outperformed CBBA in various problem instances, the PI algorithm [47] does consensus in the same way as CBBA but uses a different valuation function to compute task bids. To improve robustness to dynamic environments and robot failures, Najanath *et al.* [66] proposed an auction approach in which each task is treated separately and independently from other tasks. This method consists of executing parallel repeated auctions in which every agent is an auctioneer and bidder in parallel to enable agents to recompute task assignments.

Another class of decentralized task allocation methods consists of optimization-based approaches, divided into deterministic or stochastic optimization based approaches. Ghassemi *et al.* [68] proposed a deterministic optimization based approach in which the task allocation problem is posed as a maximum-weighted matching of a bipartite graph. An improved maximum matching algorithm is used to obtain an optimal sequence of tasks for each agent. This method assumes a deterministic environment with perfect localization and guarantees conflict-free optimal task assignments. Another deterministic optimization approach is the DHBA [44]. This method uses the Hungarian algorithm [69] to solve the task allocation problem and works in a way similar to CBAA. However, it replaces the auction phase with solving the task assignment problem via the Hungarian method on a cost matrix.

Examples of stochastic optimization-based approaches are the stochastic ant-colony optimization algorithm proposed in [67] and the decentralized GA presented in [72]. The decentralized GA was built as an extension of the GA approach presented in [73] for decentralized systems. Under full communication availability, Patel [72] showed that the decentralized GA outperformed CBBA in a number of problem instances of a rescue scenario. However, it was also shown that the performance of GA degrades significantly as communication quality decreases.

This is not an exhaustive list of task allocation algorithms available in literature. Recent works have incorporated modifications and extensions from some of the mentioned algorithms, thus we refer the reader to [74] and [75] for more comprehensive surveys of task allocation algorithms.

In previous work [32] we performed a large set of statistical tests to learn how different multi-agent task allocation algorithms perform across different communication scenarios. We leverage these results to help train our proposed meta-reasoning approach. We conduct experiments to assess the performance of the selected algorithms for these additional scenarios and identify the best performing algorithm for each scenario under various levels of communication.

We now describe each algorithm considered for the coordination of task sequence assignments in more detail.

3.1.1 CBAA

This task allocation algorithm [41] is an auction-based approach in which each agent obtains a single-task assignment. The algorithm consists of two phases: the *assignment* phase and the *consensus* phase. In the *assignment* phase, each agent computes its local bids for all incomplete tasks and assigns itself the task with the lowest bid. The agent updates its winning bids list with the lowest bid task and sends this list to all other agents. During the *consensus* phase, each agent updates its bids list with the lowest bids received and each task is assigned to the agent with the lowest bid.

3.1.2 ACBBA

This multi-task assignment algorithm [42] is an auction-based approach built as an extension of CBBA [77] for agents communicating asynchronously. This method operates in two phases: the *assignment* phase and the *consensus* phase. In the *assignment* phase, each agent gets assigned a bundle of tasks, formed by adding tasks sequentially in a greedy fashion up to the *bundle size*. Once bundles are built, agents update and share their winning bids lists along with the winning time stamps with all other agents. In the *consensus* phase, agents resolve any conflicts found on their task assignments and update their internal lists.

3.1.3 PI

The PI approach is presented in [47]. Similar to CBBA, this approach assigns multiple tasks to agents, however, it uses a different kind of bid evaluation, referred to as the "significance". This evaluation is used to assess the contribution of a task to the cost of the current task sequence assignment. There are two phases in this approach: a *task inclusion* phase and a *consensus and task removal* phase. During the *task inclusion* phase, the marginal significance of unassigned tasks is computed in order to update the task bundle and significance lists. During the *consensus and task removal* phase, each agent shares its significance list with all other agents and does consensus on the significance values received by removing tasks for which the agent has been outbid by another agent. Instead of applying CBBA consensus rules as in [47], we use ACBBA consensus rules for the consensus phase since we consider an asynchronous system.

3.1.4 DHBA

This algorithm [44] is a method that assigns a single task to each agent based on a cost matrix. This matrix is initialized using the current distance traveled and the cost of doing incomplete tasks. Thus, this matrix keeps track of the cost of each task for each agent. There are two phases in this algorithm: the *assignment* phase and the *update* phase. In the *assignment* phase, the Hungarian algorithm [78] is run on the cost matrix to obtain the optimal task assignment. In the *update* phase, agents broadcast the cost matrix and update it according to the information exchanged with other agents.

3.1.5 HIPC

This multi-task assignment algorithm [47] is built as an extension of CBBA, however, instead of generating task bundles in a greedy fashion, HIPC tries to solve the task assignment problem for all agents at once. This assignment is then used to generate bids on the tasks. This algorithm has two phases: *task allocation* phase and *consensus* phase. The *task allocation* phase consists of each agent running a full Task Allocation Algorithm (TAA) to generate its task bundle. We run a variation of the nearest neighbors algorithm [70] using the min-max objective in the TAA implementation. In the *consensus* phase, agents resolve any conflicts on task assignments using ACBBA consensus rules.

For all bundle algorithms, the current task list is reset for each agent whenever a new target is dynamically added or removed from the workspace.

In the next section, we describe the communication model used in our implementation.

3.1.6 Rayleigh Fading Model

Environmental clutter, e.g., buildings, trees, etc., tends to scatter radio signals and degrade communication quality. The propagated signals experience different shifts in amplitude, frequency and phase. The Rayleigh fading model predicts the attenuation of the received signal by assuming that the signal's amplitude will vary according to a Rayleigh distribution [48]. The Rayleigh fading model is an appropriate model to use when considering more realistic environments with many objects that can scatter the radio signal as it travels to the receiver. To quantify the fading effects, a Rayleigh random variate sequence is obtained efficiently using the Inverse Discrete Fourier Transform technique (IDFT)[79]-[80]. We sample from the generated sequence, convert the sampled power value to decibel and calculate the attenuation due to fading P_F. In our implementation, in addition to fading effects, we also account for signal attenuation due to path losses. To compute path losses P_{PL}, we use Eq. (3.1) as follows,

$$P_{PL} = P_{L_0} + 10\gamma \log_{10}\left(\frac{d}{d_0}\right), \tag{3.1}$$

where d is the current distance between the transmitter and receiver, γ is the path loss exponent and P_{L_0} is the path loss at reference distance d_0. The total attenuation is given by $P_L = P_F + P_{PL}$. The total received power is given by $P_R = P_T - P_L$, where P_T is the transmitted power. We define P_S as the user-specified sensitivity threshold. A message is dropped if the condition $P_R < P_S$ is satisfied as shown in Fig. 3.2.

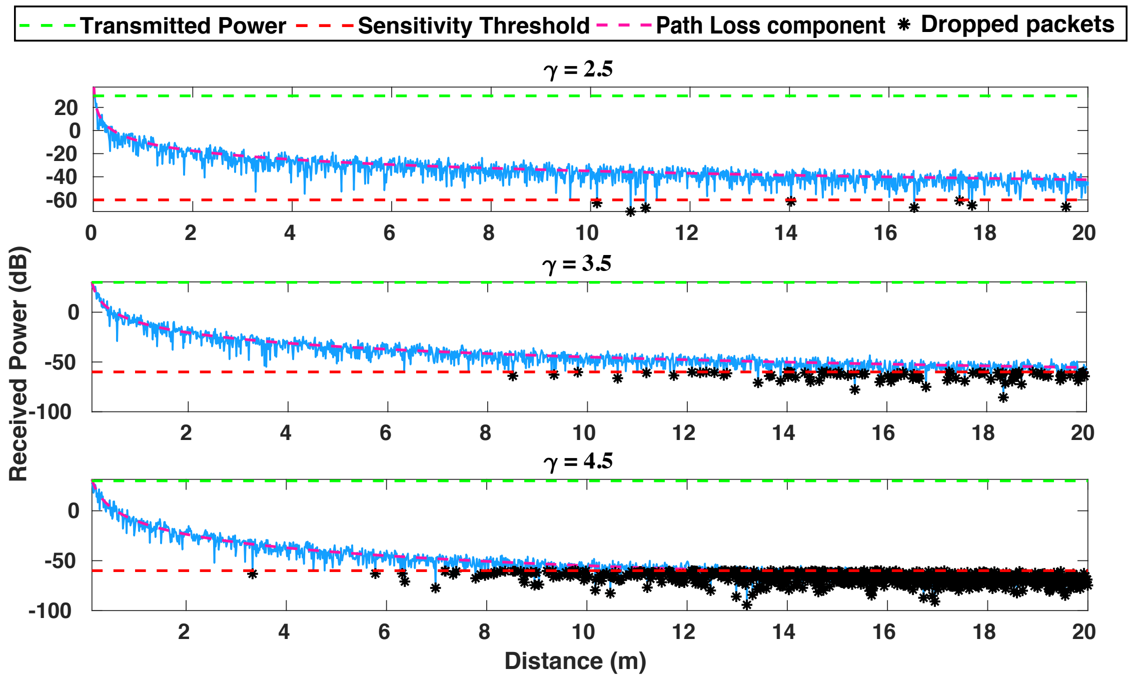


Figure 3.2: Received power of a signal attenuated by Rayleigh fading and path loss is shown for three different values of the path loss exponent γ. The transmitted power $P_T = 30\ dB$ and the sensitivity threshold $P_S = -60\ dB$. Asterisks '$*$' represent dropped packets. The number of dropped packets increases as the value of the path loss exponent increases.

In the next section, we introduce the LTL notation and the formal methods framework used for synthesizing our policy.

3.2 LTL in MAS

In recent work, the formal methods community has focused on extending hierarchical based abstractions techniques to multi-agent settings. Abstractions of dynamical systems have been extended to model MAS as finite transition systems through the use of of parallel composition [53][54] and reactive games [55]. Furthermore, the use of LTL in MAS has mainly focused on generating cooperative control strategies from rich, high-level planning objectives. Some of the objectives considered in previous works include formation and navigation [56], cooperative transportation and manipulation of objects [57], collision avoidance [58], communication [59] among others. Modeling the dynamical system of the agents is beyond the scope of this work, in which we use high-level abstractions of the communication quality in the environment and the state of the agent to formulate the synthesis problem.

In our metareasoning framework, a switching protocol between decentralized task allocation algorithms is synthesized using abstraction-based hierarchical approaches in formal methods and temporal logic planning [60][61][62][64][71]. Similar to [60], in which mode sequences are synthesized for continuous-time polynomial switched systems, we seek to determine the algorithm switching strategy that each agent must execute as to satisfy the metareasoning policy written in LTL. By expressing the policy in LTL, we can automatically generate a correct-by-construction meta-level controller. This controller can be obtained as the solution to a two-player game between the environment (communication quality) and the system (task allocation method) as is commonly done within a formal methods framework [65].

We apply reactive synthesis to generate the metareasoning policy from [83]. A benefit of using formal methods is the formal verification of our policy over an (infinite) sequence of communication states. For each action that the environment takes, a formal guarantee exists that the system will react appropriately to the environment at all times by switching to the predefined algorithm. Although a metareasoning policy could be synthesized using simpler methods, the formal framework will enable us to easily encode high-level specifications describing additional reactive behaviors of the system, laying groundwork for building extensions to this work. For example, specifications for desired behaviors may include:

1. Conditions: "If target density is perceived as high, perform a bundle-based task allocation algorithm with bundle of large size";

2. Sequencing: "first perform a multi-task assignment algorithm, then perform a single-task assignment algorithm";

3. Avoidance: "Never perform decentralized task allocation algorithm X under low communication."

In order to apply reactive synthesis, we ensure the environment is finite by discretizing the estimated communication availability into a finite number of levels.

For the system, we also obtain a finite number of states by defining the agent's object level actions as the set of all possible states.

3.3 Problem Formulation

In this section, we present the metareasoning problem solved at the meta-level layer and the task allocation problem solved at the object-level layer.

3.3.1 Metareasoning Problem

Consider a team of n agents $\mathcal{A} = \{a_1, \ldots, a_n\}$ and the multi-agent task allocation problem, denoted as $\mathcal{P}$, that $\mathcal{A}$ needs to solve. Agents can choose from a set of l multi-agent task allocation algorithms, $A = \{A_1, \ldots, A_l\}$, to solve $\mathcal{P}$. For instance, we can set $\mathcal{P} = \textit{rescue}$ and $A_1 =$ CBAA. Let $\mathcal{E}$ denote the space of environmental features such as communication level and target density. Environmental features are allowed to change as functions of time. Thus, a realization of environmental features is denoted as $e(t) \in \mathcal{E}$. Though we do not have direct access to the true value of $e(t)$ at any instant of time t, we can obtain an estimate of $e(t)$, denoted as $\tilde{e}(t)$. Consider $\tilde{e}_{i;[0:k_i]} = \{\tilde{e}_{i;0}, \tilde{e}_{i;1}, ..., \tilde{e}_{i;k_i}\}$ to be the sequence of the first $k_i + 1$ estimates computed from agent a_i. Let $\mathcal{E}_n$ be the space of all possible sequences of estimates from a team of size n. Let M be the multi-agent metareasoning approach.

Definition 4.1 *(Instantaneous Multi-Agent Metareasoning Problem):* Given $\mathcal{P}$ and a sequence of observations from all n agents, $\{\tilde{e}_{1;[0:k_1]}, \ldots, \tilde{e}_{n;[0:k_n]}\} \in \mathcal{E}_n$, where $k_i + 1$ represents the number of observations obtained by the *i-th* agent, the instantaneous multi-agent metareasoning problem is to calculate the tuples $(A_i, T_i) = M(\{\tilde{e}_{1;[0:k_1]}, \ldots, \tilde{e}_{n;[0:k_n]}\})$ composed by the multi-agent algorithm $A_i \in A$ as well as the subteam $T_i \subseteq \mathcal{A}$ that will use algorithm A_i so that the team $\mathcal{A} = \dot{\bigcup}_{i=1}^{L} T_i$, where L is the number of tuples generated, communally solves $\mathcal{P}$. Note that the subteams are disjoint sets of $\mathcal{A}$ since agents cannot belong to multiple subteams simultaneously.

Definition 4.2 *(General Multi-Agent Metareasoning Problem):* For time steps $1, \ldots, t$, repeatedly solve the Instantaneous Multi-Agent Metareasoning Problem, and then have agents in each subteam T_i use, respectively, multi-agent algorithm A_i to solve $\mathcal{P}$.

In the following section, we define the decentralized task allocation problem $\mathcal{P}$ considered for each scenario given a team of agents $\mathcal{A}$.

3.3.2 Decentralized Task Allocation Problems

The problem of decentralized task allocation can be formulated as a binary integer programming problem, similar to the multiple Traveling Salesman Problem (mTSP) [42]. Given a set of m tasks and a set of n agents, a solution is obtained such that each agent is assigned a sequence of tasks and every task in the sequence is completed by the agent. Consider a set of m tasks $\mathcal{T}$. Let $S_i \subseteq \mathcal{T}$ be a sequence of tasks assigned to agent a_i and p_i be the number of tasks in S_i. Let $q(S_i)$ be the cost of sequence S_i. The goal of the decentralized task allocation problem is to obtain a sequence S_i for each agent $a_i \in \mathcal{A}$ such that these sequences are disjoint

Task Allocation Problem	**Task Definition**	**Objective Function**
Rescue	No grid cells and known stationary targets	$\min_{\mathcal{X}} \left(\max_{a_i \in \mathcal{A}} q(S_i)\right)$
Search & Rescue	Known grid cells and unknown stationary targets	$\min_{\mathcal{X}} \left(\max_{a_i \in \mathcal{A}} q(S_i)\right)$
Fire Monitoring	Known grid cells and unknown fire targets spreading	$\min_{\mathcal{X}} \left(\max_{a_i \in \mathcal{A}} q(S_i)\right)$
Ship Protection	Known grid cells and unknown moving targets	$\max_{\mathcal{X}}(-k_0 F(\mathcal{X}) + k_1 h_1(\mathcal{X}) + k_2 h_2(\mathcal{X}))$

Table 3.1: Task allocation problem, task definition and objective function type for each scenario type. Scenarios are ordered in increasing level of difficulty.

and $\mathcal{T} = \bigcup_{i=1}^{n} S_i$. See Table 3.1. for a summary description of all the scenarios considered and Fig. 3.3, Fig. 3.4, Fig. 3.5, Fig. 3.6 for examples of the simulation runs for each type of scenario.

We define $\mathcal{T}$ and $q(S_i)$ for each type of scenario as follows:

3.3.2.1 Rescue Scenario

In the rescue scenario, $\mathcal{T}$ is defined as a finite set of *a priori* known stationary targets $\mathcal{U} = \{u_1, \ldots, u_m\}$ located in a map, $\mathcal{W} \subset \mathbb{R}^2$, of size $N \times N$. We define $\mathcal{T} \triangleq \mathcal{U}$ and S_i to be the sequence of p_i targets to be visited by agent a_i. A target is considered to be visited when an agent moves within a threshold distance δ_T of the target's location. The mission in this scenario is completed when every target has been visited by at least one agent.

The *min-max* objective $\mathcal{O}(\mathcal{X})$ considered in this scenario is to find a task assignment $\mathcal{X}^* = \{S_1, \ldots, S_n\}$ such that

$$\mathcal{O}(\mathcal{X}^*) = \min_{\mathcal{X}} \left(\max_{a_i \in \mathcal{A}} q(S_i) \right). \tag{3.2}$$

The cost function $q(S_i)$ is defined as

$$q(S_i) = C_i + c_i(u_1) + \sum_{k=1}^{p_i - 1} \|u_{k+1} - u_k\|, \tag{3.3}$$

where C_i is the cost accrued by a_i up to its current location and $c_i(u_k)$ corresponds to a_i's cost of visiting target u_k. The cost $c_i(u_k)$ is calculated as the Euclidean distance from agent a_i to target u_k. Thus, the min-max objective is to minimize the maximum distance traveled over all agents. Assuming a constant speed for the agents, this is equivalent to minimizing the mission completion time.

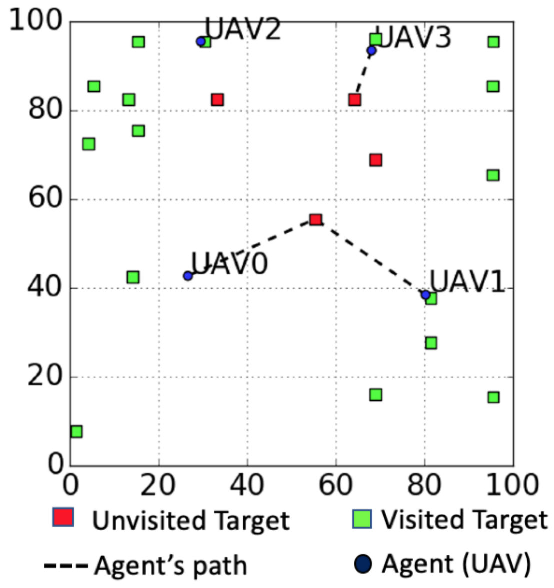


Figure 3.3: Possible runs of agents performing decentralized TA for the rescue scenario. Each target has a unique ID, not shown to reduce clutter.

3.3.2.2 Search & Rescue Scenario

In this scenario, we define $\mathcal{T} \triangleq \mathcal{G} \cup \mathcal{U}$, where $\mathcal{G} = \{g_1, \ldots, g_r\} \subset \mathcal{W}$ is a finite set of *a priori* known grid cells and $\mathcal{U} = \{u_1, \ldots, u_m\} \subset \mathcal{W}$ is a finite set of unknown stationary targets. The cells in $\mathcal{G}$ divide the search space into regions of equal size. Initially, agents begin searching the map by visiting each cell in $\mathcal{G}$. Each cell is said to be completely searched when an agent reaches its center because, at this location, an agent's sensor radius R_d covers the entire cell, and the agent can detect any targets in that cell. As agents search the space, they are able to detect new targets located within their sensor radius. Discovered targets are added to the set of known tasks $\mathcal{K}$, which is equal to $\mathcal{G}$ at the start of the mission. Agents share information about the newly discovered targets with other agents. Thus, S_i is the sequence of p_i tasks (stationary targets and grid cells) in $\mathcal{K}$ that are assigned to a_i. The mission is completed when every cell is searched by at least one agent and every target is visited by at least one agent.

The *min-max* objective $\mathcal{O}(\mathcal{X})$ and cost function $q(S_i)$ in this scenario are defined in the same way as in the rescue scenario.

3.3.2.3 Fire Monitoring

We define $\mathcal{T} \triangleq \mathcal{G} \cup \mathcal{U}$ as in the search & rescue scenario with the difference that stationary targets in this scenario correspond to fire locations which are dynamically generated and added to the map during the mission. Fire locations are propagated according to the wavelet differential equations and the Rothermel spread equation

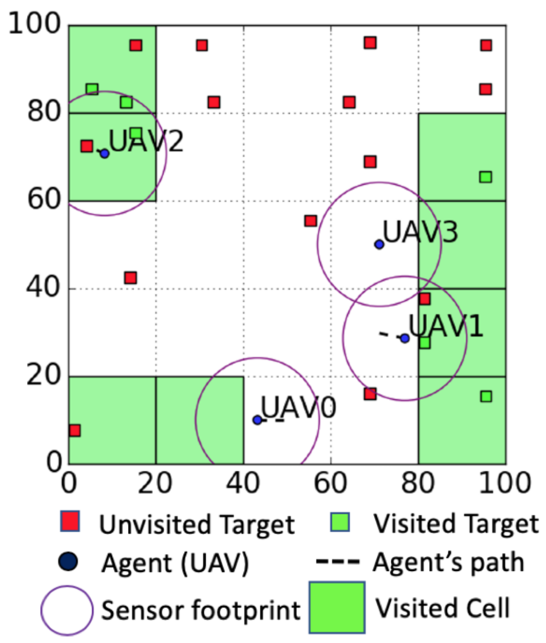


Figure 3.4: Possible runs of agents performing decentralized TA for the search & rescue scenario.

provided in Farsite [21], a fire simulation engine. At each simulation step, fire grows continuously for a time interval Δt, after which new fire locations are discretized into fire regions. Each fire region obtained is considered as an stationary target. Fire locations are propagated until a max number of targets is reached. Discovered fire targets are added to the set of known tasks $\mathcal{K}$, which is equal to $\mathcal{G}$ initially. Agents share locations of the newly discovered fire targets with other agents. Thus, S_i is the sequence of p_i tasks (fire targets and grid cells) in $\mathcal{K}$ that are assigned to a_i. The mission is completed when when every cell has been search by at least one agent and every fire target has been visited by at least one agent.

The *min-max* objective $\mathcal{O}(\mathcal{X})$ and cost function $q(S_i)$ in this scenario are defined in the same way as in the rescue scenario.

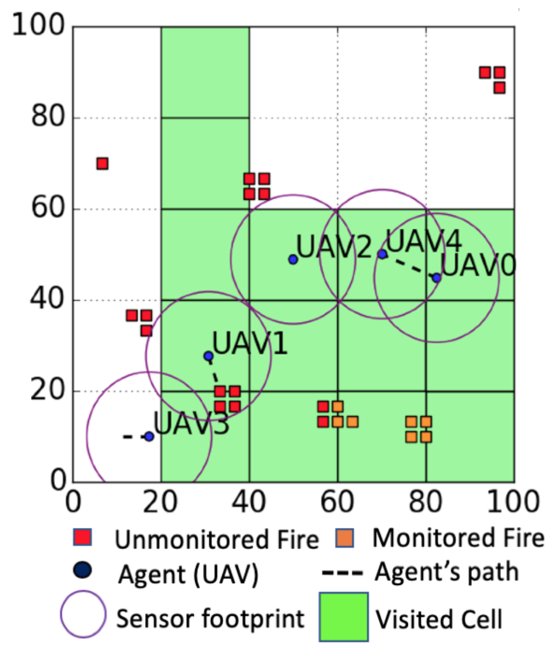


Figure 3.5: Possible runs of agents performing decentralized TA for the fire monitoring scenario.

3.3.2.4 Ship Protection

In this scenario, we consider a ship with location (s_x, s_y), initially placed at the boundary of the map. During the mission, the ship moves with a constant speed v_s and constant heading θ_s to the opposite side of the map. The tasks in this scenario correspond to the set $\mathcal{T} \triangleq \mathcal{G} \cup \mathcal{U}$, where $\mathcal{G} = \{g_1, \dots, g_r\} \subset \mathcal{W}$ is a finite set of *a priori* known grid cells and $\mathcal{U} = \{u_1, \dots, u_n\} \subset \mathcal{W}$ is a finite set of unknown moving targets. Unlike all previous scenarios, cells here are searched continuously since targets can move across different cells and might appear in a cell already searched. Each agent knows, for each cell g_i, the elapsed time t_{g_i} since the last time that some agent searched that cell. A cell is said to be searched when an agent reaches the center of the cell, at which point t_{g_i} is reset to 0. Agents broadcast the locations of all newly discovered targets in the cells and the timestamps at which the cells were last searched.

As agents search the space, they can detect, classify and track, if necessary, all moving targets in the map in order to protect the ship. There are two types of moving targets in the set $\mathcal{U}$: adversarial and non-adversarial. Adversarial targets move towards the ship using a directed random walk while non-adversarial targets move randomly in the map. All targets move along piece-wise linear trajectories and remain within a threshold distance δ_s from the ship. Let θ_{u_i} be the heading of each adversarial target $u_i \in \mathcal{U}$. This heading is changed randomly at fixed intervals of time, otherwise it is computed using the current location of the ship (s_x, s_y) and the location of the target (u_{ix}, u_{iy}) as

$$\theta_{u_i} = \tan^{-1}(\frac{s_y - u_{iy}}{s_x - u_{ix}}). \tag{3.4}$$

The heading for each target is sampled from $[-90°, 90°]$ and the target speed can vary between 0 and v_u, which is set to be strictly less than the agents' max speed. We do this to ensure that the agents succeed in tracking all the targets.

In addition to the sensor radius R_d that each agent uses for detecting targets, we define a classification radius R_c. When a target moves within an agent's classification radius, it can be classified as adversarial or non-adversarial by the agent. We set $R_c < R_d$ in order to move agents closer to targets in their effort to classify them. If the target is found to be adversarial, the agent proceeds to track it by moving towards the target using a proportional controller with control law,

$$w = k_p(x_{ui} - x_{ai}) + v_{ui}, \tag{3.5}$$

where k_p is the controller gain, x_{ui} is the current position of target u_i, x_{ai} is the current position of agent a_i, and v_{ui} is the current velocity of u_i. An adversarial target is considered to be tracked when an agent moves within a threshold distance δ_T of the target's location. A tracked adversarial target will move away from the ship towards the boundary of the map and eventually leave the map for the rest of the mission. A non-adversarial target successfully classified by an agent is considered to be tracked immediately after. A tracked non-adversarial target will continue to move along its trajectory unaffected by the agent's actions.

Newly discovered targets are added to the set of known tasks $\mathcal{K}$, initially set equal to $\mathcal{G}$. Newly tracked targets are added to the set $\mathcal{Z} \subseteq \mathcal{U}$, which is empty at the start of the mission. Thus, S_i is the sequence of p_i tasks (grid cells and moving targets) in $\mathcal{K}$ that are assigned to a_i. The mission is finished when the ship reaches the opposite side of the map.

The *max* objective $\mathcal{O}(\mathcal{X})$ considered in this scenario is to find a task assignment $\mathcal{X}^* = \{S_1, \ldots, S_n\}$ such that

$$\mathcal{O}(\mathcal{X}^*) = \max_{\mathcal{X}} -k_0 F(\mathcal{X}) + k_1 h_1(\mathcal{X}) + k_2 h_2(\mathcal{X}), \tag{3.6}$$

where $F(\mathcal{X})$ is the max travel cost to be accrued by each agent $a_i \in \mathcal{A}$ performing its assigned task sequence S_i over all agents. This term is weighted by k_0, a user-specified value. We define this term as

$$F(\mathcal{X}) = \max_{a_i \in \mathcal{A}} q(S_i). \tag{3.7}$$

The cost function $q(S_i)$ is defined as in Eq. (6) where $c_i(u_k)$ corresponds to agent a_i's cost of tracking a moving target or visiting a grid cell in $\mathcal{K}$.

The second term in the objective is the expected distance to the ship over all unassigned cells and is defined as

$$h_1(\mathcal{X}) = \min_{g \in \mathcal{G} \setminus \bigcup_{a_i \in \mathcal{A}} S_i} p^{t_g} d_s(g), \tag{3.8}$$

where p is a value in $(0, 1)$, t_g is the time elapsed since cell g was searched and $d_s(g)$ is the distance between the center of the cell and the location of the ship. The effect of $h_1(\mathcal{X})$ is to prioritize searching unassigned cells that are closer to the ship and that have not been visited for a longer time.

The third term in the objective is the minimal distance to the ship over all untracked targets and is defined as

$$h_2(\mathcal{X}) = \min_{u \in \mathcal{K} \setminus \mathcal{Z}} d_s(u), \tag{3.9}$$

where $d_s(u)$ is the distance between the target's location and the ship. The effect of $h_2(\mathcal{X})$ is to prioritize tracking targets that are closer to the ship. The terms $h_1(\mathcal{X})$ and $h_2(\mathcal{X})$ are weighted by k_1 and k_2, respectively, both of which are user-defined coefficients. Thus, the max objective is equivalent to a weighted combination of minimizing the maximum distance traveled over all agents, maximizing the minimum expected distance to the ship over all unassigned cells and maximizing the minimum distance to the ship over all untracked targets.

A second objective that we consider across all scenarios is the min-max number of transmitted messages, whether received or not. Both performance metrics, max travel distance (equivalent to mission time) and number of transmitted messages, are critical in a number of applications in which agents may experience limited communication such as area coverage [84], environmental monitoring [85], emergency management [86], and search & rescue missions [87].

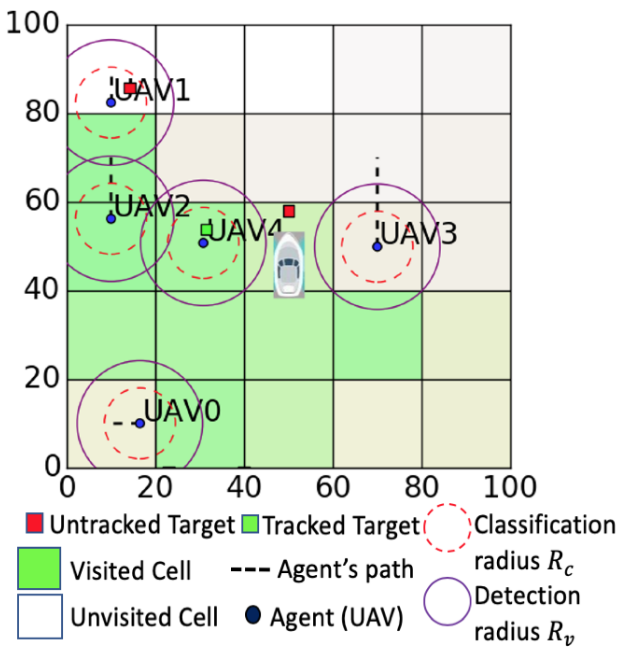


Figure 3.6: Possible runs of agents performing decentralized TA for the ship protection scenario. In this scenario, lighter shades of green reflect that a longer time has passed since cells were last searched.

3.4 Metareasoning Framework

Our metareasoning framework defines three control layers (shown in Fig. 3.7) in every agent:

(i) *Meta-level layer*: This layer decides on the decentralized task allocation algorithm the agent should perform (meta-level action).

(ii) *Task-planning layer*: This layer decides on the task sequence assignment (object-level action).

(iii) *Low-level layer*: This layer generates the trajectories along which the agent needs to move to reach each task location.

Each agent's decision cycle consists of:

(a) Estimating the communication level in the environment at the task-planning layer from heart-beat messages received at the low-level layer.

(b) Executing meta-level control to output the appropriate algorithm for the perceived communication level according to the metareasoning policy.

(c) Performing the chosen algorithm to obtain a task sequence assignment.

The novelty of our approach lies in the policy we place in the meta-level control layer. This layer consists of a switching protocol as defined by a fixed, common metareasoning policy computed offline. This ensures that agents perform the same algorithm for a given level of communication without the need to communicate or

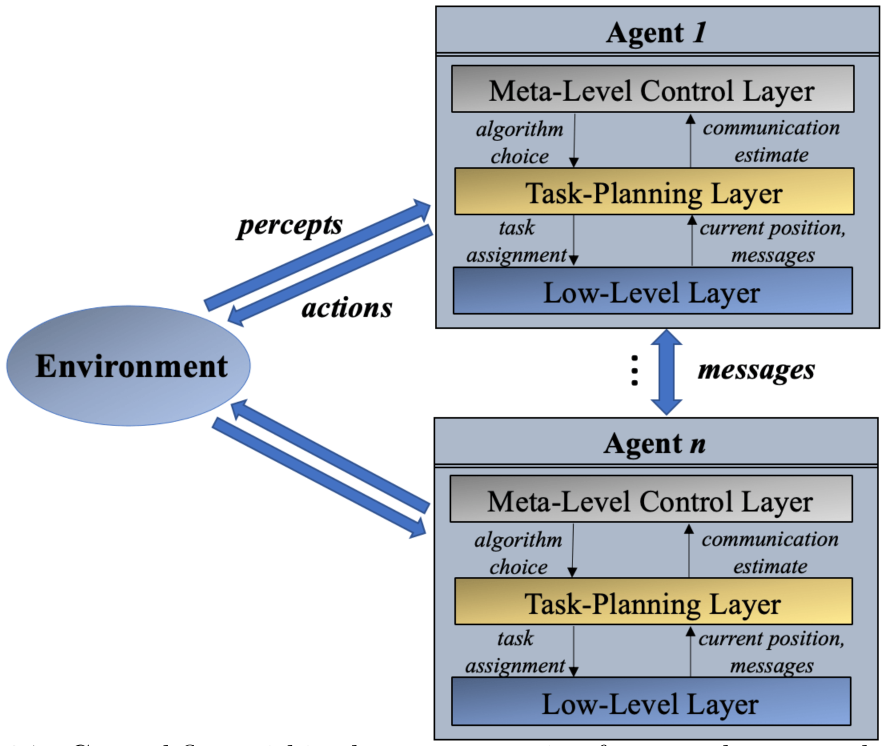


Figure 3.7: Control flow within the metareasoning framework proposed. Each agent has a meta-level control layer, a task-planning layer and a low-level layer. Agents compute a communication estimate from the messages received at the low-level layer. Using this estimate, the meta-level control layer outputs the algorithm choice for the agent.

synchronize their decisions during runtime. Since agents compute communication quality estimates locally and run an independent copy of the policy, different agents may have different beliefs about the communication quality at any instant of time and therefore may use different algorithms simultaneously.

We synthesize the metareasoning policy as the solution to the reactive synthesis problem involving the communication level in the environment and the agent's algorithm choice. In the following sections, we define the assumptions on the environment and the system requirements.

3.4.1 Environment

Each agent sends a fixed number of heartbeat messages periodically for the purpose of communication estimation. These heartbeat messages do not contain any meaningful information, but their absence can signal loss of communication between a sender and a receiver. Each agent a_i estimates the per link communication with agent a_j by computing the ratio of the number of heartbeat messages received per link h_{ij} and the expected number of heartbeat messages h. At each time step, agent a_i can determine its communication estimate c_{e_i} by computing the max ratio over all its communication links,

$$c_{e_i} = \max_{a_i \neq a_j, a_j \in \mathcal{A}} (\frac{h_{ij}}{h}). \tag{3.10}$$

The communication estimate c_{e_i} is then mapped to a discrete communication level. Nayak et al. [32] showed that the performance ranking of the task allocation algorithms tested remains the same at high communication levels for the visit and search & visit scenarios. It is only when communication drops substantially that this ranking changes. For this reason, we define only two discrete communication levels based on the analysis presented in [32]: high (H) for $c_t < c_{e_i} \leq 1$ and low (L) for $0 \leq c_{e_i} \leq c_t$. The threshold value c_t is specified experimentally by mapping the sensitivity threshold values tested in [32] to either high or low communication based on the sensitivity threshold value at which the change in ranking of the algorithms was observed. In the synthesis problem, each of these discrete communication levels is represented by an environment variable, which is defined as a Boolean. Thus, the environment in our problem can be described by these two variables as $E = H \times L$. This implies that the state of the environment corresponds to a valuation of these two Boolean variables. If the communication estimate c_{e_i} is within the range $c_t < c_{e_i} \leq 1$, then H will be set to $True$ and L will be set to $False$. Otherwise, if $0 \leq c_{e_i} \leq c_t$, H will be set to $False$ and L will be set to $True$. The environment specifications are as follows:

$$\varphi_i^e = L \wedge \neg H, \tag{3.11}$$

$$\varphi_t^e = \square(\neg(H \wedge L) \wedge (H \vee L)), \tag{3.12}$$

$$\varphi_g^e = \square\Diamond H \wedge \square\Diamond L. \tag{3.13}$$

Eq. (3.11) states that initially the environment is assumed to have a low communication level. This assumption is valid if we expect it will take a few milliseconds for all agents to start up the exchange of messages. Though, any initial condition can be chosen as long as it satisfies all other specifications. Eq. (3.12) is required to verify that exactly one of the two variables H and L is true at all times. This is specified to ensure that the environment is in a single level of communication, ignoring cases in which communication is simultaneously high and low or simultaneously neither. Finally, Eq. (3.13) states that always eventually: the communication level of the environment will be high and that always eventually: the communication level of the environment will be low. This specification ensures that the communication in the environment will change at some future time. Communication changes may occur due to weather conditions, clutter in the environment, dynamic obstacles, or even adversarial agents that seek to interfere communication among agents.

3.4.2 System

Let $S = \{A_j, A_k\} \times \{True, False\}$ be the set of states for each agent, where $\{A_j, A_k\} \subset A$. An element $s \in S$ represents the tuple composed by the multi-agent task allocation algorithm used by the agent and a Boolean variable, denoted as $Reset$. This variable will be set to $True$ when the agent's current task sequence assignment and bids need to be reset before executing the task allocation process. If $Reset$ is set to $False$, then the agent performs the task allocation process using its current task sequence assignment and winning bids information. The rationale for this is that as communication improves, agents are likely to obtain better solutions and may benefit from discarding solutions of lower quality obtained under poor communication.

Let $F \subseteq S \times S$ be the set of all possible transitions between the elements in the state space. Thus, the agent can transition between any two task allocation algorithms. When switching happens, agents only need to transfer their current winning bids, completed tasks and current sequence assignment of tasks into the execution of an iteration of the new algorithm.

We define the following system specifications:

$$\varphi_i^s = A_k \ \wedge \ Reset, \tag{3.14}$$

$$\varphi^s_{t,1} = \square((L \wedge \bigcirc H) \rightarrow \bigcirc(A_j \wedge \mathit{Reset})), \tag{3.15}$$

$$\varphi^s_{t,2} = \square((H \wedge \bigcirc L) \rightarrow \bigcirc(A_k \wedge \neg\mathit{Reset})), \tag{3.16}$$

$$\varphi^s_{t,3} = \square((L \wedge \bigcirc L) \rightarrow \bigcirc(A_k \wedge \neg\mathit{Reset})), \tag{3.17}$$

$$\varphi^s_{t,4} = \square((H \wedge \bigcirc H) \rightarrow \bigcirc(A_j \wedge \neg\mathit{Reset})). \tag{3.18}$$

Eq. (3.14) states that agents reset their task sequence assignments and perform A_k initially. Eqs. (3.15)-(3.18) indicate which algorithm the agent should perform and whether or not the agent should reset its current task sequence assignment next based on the current and next states of the environment. Eq. (3.15) specifies that, when communication improves, agents should reset their task sequence assignments. Eqs. (3.16)-(3.18) specify that, when communication degrades or remains the same, agents should perform the task allocation process specified using their current task sequence and bids information.

All scenarios use the specifications as described in Eqs. (3.14)-(3.18). For each scenario, we identify the best performing algorithms for high and low communication levels, respectively, via simulation experiments. We assign the best performing algorithm under high communication to A_j and the best performing algorithm under low communication to A_k. For the rescue and search & rescue scenarios, we leverage the results from [32] and set A_j = ACBBA and A_k = CBAA. We perform further experiments to set A_j and A_k for the fire monitoring and ship protection scenarios.

3.4.3 Alternative Team Up Strategy

When agents switch to performing different algorithms simultaneously because of having different beliefs about the communication quality in the environment, they may compute task assignments from outdated information. This information may have been successfully received at some point in time, but may no longer be valid once the agents lose communication and change their task allocation schemes. To address some potential conflicts from changes in communication, we propose a *team up* strategy in which agents fully collaborate only with agents from which they perceive a high level of communication. Hence, if an agent *teams up* with another agent, it will use all the bids information received from the agent when executing task allocation. On the contrary, if an agent perceives low communication from another agent, it will discard all bids information received from this agent except for information pertaining completed tasks. Fig. 3.8 shows agents a_0, a_1 and a_2 performing the *team up* strategy.

The motivation for this strategy is to prevent agents from relying on all the information received from agents with which they experience poor communication. Consider an agent a_1 that receives messages about tasks that another agent a_2 intends to do and plans accordingly. However, if a_2 decides to do different tasks and broadcasts messages about its updated plan, these messages may not be received by a_1. This could lead to a_1 and a_2 leaving some tasks incomplete or taking much longer to complete all tasks. By enabling agents to select which information to include in their computations based on the perceived communication quality, tasks can be completed even when all broadcasted messages are not received. We test this *team up* strategy when we perform the metareasoning experiments.

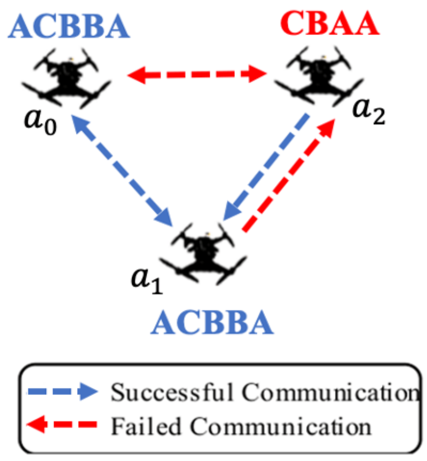


Figure 3.8: Sample execution of the synthesized switching protocol where agents team up according to the level of communication perceived. Agents a_0 and a_1 team up as they perform ACBBA, while a_2 individually performs CBAA.

3.5 Experimental Setup

The simulation framework is built in the Robot Operating System (ROS) [88] Kinetic. Two types of modules are implemented: agent and environment. The task allocation algorithms, written in Python, are executed in the processing unit of each agent module every 0.1s. Once a task assignment is obtained, the agent starts moving towards the task location unless it receives a message from another agent indicating that the task has already been completed. Only when the agent arrives at its assigned task will it proceed to complete its next assigned task. The agents communicate their solutions and heartbeat messages over the ROS network through a communication interface written in C++. Separate processes for all agents are used in order to simulate a decentralized system. Only one process is used for the environment module. The environment module simulates the agents as point robots as well as the 2D map in which the agents move. We assume a collision free model for the agents. To move in the map, the agent module sends a request to the environment simulator, which provides odometry sensor readings for the agent. For more details about this simulation framework, see [32].

We ran all simulations on an on an AMD Ryzen Threadripper 2990WX. In all scenarios except for the ship protection scenario, simulations were terminated when at least one agent came to know all target locations had been visited. In the ship protection scenario, simulations were terminated when the ship arrives to the other side of the map.

Regarding the communication topology, we assume a *full mesh* topology in which every agent attempts to communicate with every other agent by continuously broadcasting messages. The Rayleigh Fading model is used to determine whether or not a message is dropped. Thus, the topology of the network is sparse and changes as a function of space and time.

3.5.1 Design of Experiments

For all scenarios, we used a randomized design of experiments. The dimension of the map is $N \times N$ with $N = 100$. We set the agent speed $A_s = 6$ units/s for all scenarios except the ship protection scenario. In this scenario, we used a proportional controller and set the max $A_s = 12$ units/s so that agents can track targets before they move out of sensor range. The map is divided into 25 grid cells of size 20×20. The detection radius $R_d = 14.14$ units (half length of diagonal of grid cell) while the classification radius $R_c = 8$ units. We set the threshold distance $\delta_T = 0.25$ units.

For the rescue, search & rescue and ship protection scenarios, target locations were sampled from a 2D Gaussian Mixture Model described as follows,

$$p_{gmm} = \frac{1}{K}\sum_{i=1}^{K} \mathcal{N}(\mu_i, \Sigma_i), \tag{3.19}$$

where K is the number of clusters, each with radius r_i, center at μ_i and co-variance

$$\Sigma_i = \begin{pmatrix} r_i^2 & 0 \\ 0 & r_i^2 \end{pmatrix}. \tag{3.20}$$

For the fire monitoring scenario, initial fire locations were uniformly sampled from $[0, 100] \times [0, 100]$. We set additional parameters including wind speed and wind direction.

For the ship protection scenario, we set the initial location of the ship to (50, 0), heading $\theta_s = 90°$and speed $v_s = 1.6$ units/sec. In addition, we set the ratio of number of adversarial targets to non-adversarial targets and exclude initial locations of targets that have a distance < 40 units from the ship. We set the default speed for all moving targets as 4 units/sec. If adversarial targets are tracked, their speed is set to 12 units/sec.

For all experiments, we set the Rayleigh fading model parameters as N= 64, d_0=1m, P_T= 30 dB, and $P_{L_0} = 40\ dB$ as suggested in [32].

An instance is defined as one random sampling of the parameters from the ranges mentioned in Table 3.2. for the rescue, search & rescue and ship protection scenarios. For the fire monitoring scenario, we used the parameters from the ranges indicated in Table 3.3.

Parameter	**Range**
Number of agents	[5,10]
Number of target clusters (K)	[1,4]
Ratio of number of targets to agents	[1,4]
Initial locations of agents	([0, 100],[0, 100])
Cluster centers (μ_i)	([0, 100],[0, 100])
Cluster radii (r)	[15,50]
Ratio of number of adversarial targets to non-adversarial targets *	[1,3]

Table 3.2: Parameter ranges for instance generation in the rescue, search & rescue and ship protection scenarios. The * indicates parameter applies only to the ship protection scenario.

Parameter	**Range**
Number of agents	[5,10]
Number of initial fires	[4,10]
Initial locations of fires	([0, 100],[0, 100])
Initial locations of agents	([0, 100],[0, 100])
Wind speed (units/sec)	[5,20]
Wind direction (radians)	$[0,2\pi)$

Table 3.3: Parameter ranges for instance generation in fire monitoring scenario.

3.5.2 Determination of weighting coefficients for ship protection scenario

Unlike the other scenarios, the ship protection scenario uses a multi-objective function (3.6) to minimize max distance traveled while driving agents to track targets closer to the ship and continuously search cells in the workspace. The three terms in the objective function are weighted by k_0, k_1 and k_2, respectively. Changing these weights affects the agents' task allocations since task bids are computed using this multi-objective function for all algorithms. Hence, for each algorithm we ran experiments to determine the weights that exhibit the best performance on average with respect to the minimum distance to the ship amongst all targets. We chose this metric since protecting the ship is the primary goal in this scenario.

Assuming perfect communication, we generated 40 samples for (k_0, k_1) using the Latin Hypercube Sampling method [90]. We sampled k_0 and k_1 from $[10, 1000]$. This range was determined empirically by analyzing the agents' behavior at extreme values of k_0 and k_1 and verifying that their behavior was different. At $k_0 = 10$ and $k_1 = 1000$, agents often visited cells closer to the ship even when they were far away from these cells. At $k_0 = 1000$ and $k_1 = 10$, agents were often assigned to visiting cells that were closer to their own locations. We also observed that $k_2 = 500$ was an effective value to make agents prioritize tracking targets for all values of (k_0, k_1) tested. We performed 50 experiments with 5 agents and 20 targets for each generated sample of (k_0, k_1) and for each of algorithm. Initial locations of agents and targets were randomly generated. We set the number of adversarial targets equal to the number of non-adversarial targets.

We evaluate the results by calculating the average and 95 percent confidence intervals of the chosen metric across the 50 experiments. Amongst the 40 samples, we selected the 3 candidate samples with the highest average minimum distance to the ship. From the selected samples, we kept the one with the smallest confidence interval as this indicated more consistency in performance across scenarios. Table

3.4. shows the best performing weights on average for all algorithms.

	Weights		
Algorithm	k_0	k_1	k_2
CBAA	621	655	500
DHBA	858	258	500
HIPC	109	478	500
ACBBA	483	320	500
PI	215	248	500

Table 3.4: Best performing weights on average for the ship protection objective function with respect to minimum distance to the ship.

3.5.3 Determination of optimal algorithm parameters for fire monitoring and ship protection scenarios

The parameter space of CBAA, DHBA and HIPC contains the max iteration count $\mathcal{I}$, and for ACBBA and PI, it contains both $\mathcal{I}$ and max bundle size $\mathcal{B}$. Values for the $\mathcal{I}$ and $\mathcal{B}$ tuning parameters are chosen as follows: for each $\mathcal{I} \in \{1, 2, 3, 4, 5, 10, 15, 20\}$ and $\mathcal{B} \in \{2, 3, 4, 5, 10, 20, 30, 40\}$. The tuning was done assuming perfect communication. For all the experiments, we used 7 agents, 22 targets for the ship protection scenario and a max of 40 fire targets for the fire monitoring scenario. We set the number of adversarial targets equal to the number of non-adversarial targets in the ship protection scenario.

We performed 15 experiments for each point in the parameter space, with each experiment having randomly determined starting locations of targets and agents. The performance metrics across all the experiments were averaged for each point in the parameter space. For each algorithm, we selected the point in the parameter space that had the least average max distance traveled. The distribution of this point was compared with the distributions of all other points in the parameter space using the Wilcoxon-Signed Rank (WSR) test [91]. If the distributions were similar with a 95 percent confidence and the difference in the average max distance traveled was less than 15 percent, we chose the point with the lowest number of transmitted messages.

The plots for the ship protection scenario and the fire monitoring scenario results are shown in Fig. 3.9. Table 3.5. shows the best parameters for each of these scenarios and for each algorithm. For the rescue and search & rescue scenarios, we used the iteration counts and bundle sizes suggested in [32].

	Algorithm				
Scenario	CBAA	DHBA	HIPC	ACBBA	PI
Fire monitoring	2	2	1	(1, 40)	(1, 2)
Ship protection	4	1	1	(1, 40)	(1, 2)

Table 3.5: Best algorithm parameters for each algorithm: $\mathcal{I}$ (iteration count) or ($\mathcal{I}$ $\mathcal{B}$) (iteration count, bundle size).

3.5.4 Design of Comparison Experiments

To compare the performance of the 5 algorithms for the ship protection and fire monitoring scenarios, we used a similar design of experiments and analysis as described in [32]. For each type of scenario, we generated 50 instances with different parameter values mentioned in Table 3.2. and Table 3.3. respectively. With regards to communication, we varied the sensitivity threshold P_S in the range [-25, -75] dB in -10 dB increments. Every instance was run at each sensitivity threshold. We used the WSR test and Metrics Trade-off plots (MTP) to evaluate the performance of the algorithms in each scenario.

Observing the MTP for the fire monitoring scenario, we note that ACBBA, HIPC, DHBA and CBAA consistently appear on the plots. From the WSR test, it can be inferred that ACBBA and HIPC have no statistical difference in either performance metric, thus implying no trade-off. ACBBA is chosen amongst this pair as it minimizes the max distance traveled. DHBA and ACBBA exhibit statistical difference in messages transmitted and max distance traveled at the highest level of communication. Therefore, we again choose ACBBA for high communication since it minimizes the max distance traveled. ACBBA and CBAA exhibit trade-off at most communication levels. As we prioritize minimizing max distance traveled, we choose ACBBA at high communication levels. At low communication levels, we note that ACBBA and CBAA have no statistical difference in max distance traveled. This is similar to PI and CBAA, which do not exhibit statistical difference in either metric at the lowest level of communication. We can choose CBAA or PI over ACBBA as both minimize the number of messages transmitted. Thus, for low communication, we choose CBAA since it achieves the lowest number of messages transmitted.

Performing a similar analysis on the ship protection scenario, we note that PI, ACBBA and DHBA consistently appear on the trade-off plot. At the highest communication level, we note that PI and ACBBA have statistical difference in max distance traveled and number of messages transmitted which implies that a trade-off exists. Prioritizing the minimum distance to ship metric, we choose PI over ACBBA.

PI and DHBA do not exhibit statistical difference in max distance traveled but PI obtains a higher mean for minimum distance to ship, thus we choose PI. At the lowest level of communication, we again choose DHBA over ACBBA despite there being statistical difference in both max distance traveled and number of messages. This is done to prioritize minimum distance to ship in which DHBA performs better. Fig. 3.11 shows the trade-off and Wilcoxon Signed Rank test results for the fire monitoring and ship protection scenarios.

3.5.5 Determination of communication threshold value

To identify the value of c_t, we ran 15 experiments using the rescue scenario with 25 targets for simplicity and 2 agents for a single communication link. Since the communication estimate is independent of the type of scenario, we limited the experiments to the rescue scenario. Agent and target starting locations were randomly generated. We varied the sensitivity threshold values from -25 dB to -75 dB in increments of -10 dB. We computed the average over each agent's communication estimates obtained over all time steps. We calculated the final communication estimate for the experiment by taking the average over the estimates obtained from all the agents. We then computed the mean estimate from all 15 experiments for each sensitivity threshold value. Fig. 3.12 shows that sensitivity threshold values above -35 dB corresponded to mean communication estimates below 0.2. Thus, c_t was set to 0.2.

To simulate different levels of communication, we first determined the ranges of values for γ corresponding to high and low levels of communication, respectively, based on c_t. Generally, values of γ range from 2 (less cluttered environments) to 5 (more obstructed areas) [89]. Thus, we ran the same 15 experiments described above with the difference that we set P_S to –65 dB and instead varied γ from 2 to 5 in increments of 0.25. We used these experiments to identify the ranges of values for γ that would likely result in communication estimates above and below c_t. Fig. 3.13 shows that values from 2.0 to 3.0 resulted in mean communication estimates above c_t while values from 4.5 to 5.0 resulted in mean communication estimates below c_t.

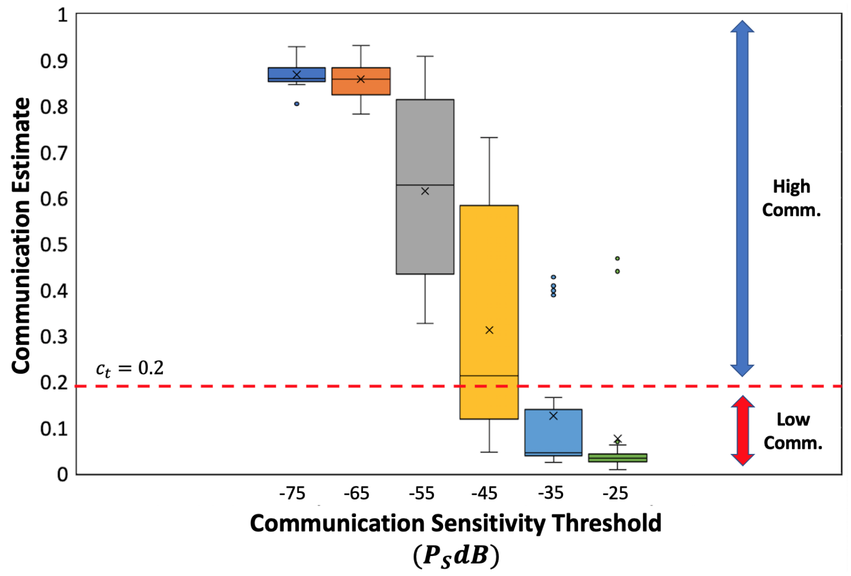


Figure 3.12: Communication estimate values are shown for different sensitivity threshold values using the Rayleigh Fading model. Segmented line indicates the communication estimate value above which we define as high communication.

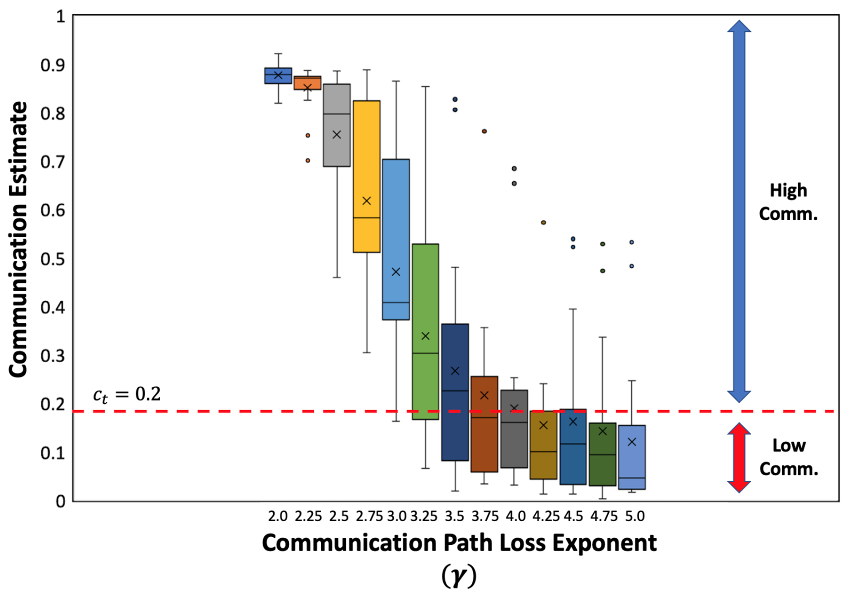


Figure 3.13: Communication estimate values are shown for different path loss exponent values using the Rayleigh Fading model. Segmented line indicates the communication estimate threshold value used to define high and low communication.

3.5.6 Design of Metareasoning Experiments

To test the metareasoning policy, we first identified the best performing algorithms under high and low communication levels for each type of scenario. For the rescue and search & rescue scenarios, we selected the best performing algorithms based on the results from [32]. For the fire monitoring and ship protection scenarios, we ran experiments to characterize the performance of each of the 5 algorithms considered under different levels of communication. These experiments were performed using the optimal algorithm parameters for each scenario and the optimal weighting coefficients k_0, k_1 and k_2 in the objective function for the ship protection scenario.

Using the Rayleigh Fading model, results from [32] showed that the performance ranking of the task allocation algorithms tested changed at a sensitivity threshold value of -25 dB for the visit scenario and -35 dB for the search & visit scenario. Thus, we define values below -35 dB as high communication levels and values above -35 dB as low communication levels. For each sensitivity threshold value tested in [32], we ran experiments to compute the corresponding communication estimate. We then set the value of the communication threshold c_t to the communication estimate at which communication levels change from low to high.

Theoretically, we could vary the sensitivity threshold values to simulate high and low communication. However, this would not be feasible in a hardware implementation since the sensitivity threshold is inherent to the agent. Instead, we choose to vary the path loss exponent γ, which is equivalent to generating different amounts of clutter in the environment [89]. We used the values of c_t and the ranges of γ values found in previous section.

We selected the algorithms for the metareasoning policy for each type of scenario from the results obtained. See Table 3.6. for a summary of these results. To demonstrate the performance of our metareasoning approach, we ran experiments for all the scenario types considered. Our policy as well as each algorithm used in the policy were tested on multiple instances of each scenario type at two different switching conditions of communication: low to high and high to low. These conditions represent possible ways in which communication may change during a mission. We sampled values for γ from $[2.0, 3.0]$ to simulate high communication and from $[4.5, 5.0]$ to simulate low communication. We set the switching time for communication t_1 as a random value in $[5, 15]$. The heartbeat rate is set to 5 messages per second.

A total number of 50 instances was generated for each type of scenario and communication switching condition. The total number of experiments came to be $50 \times 2 \times 4 = 400$. We tested the metareasoning policy as well as each algorithm used in the policy across all these experiments in order to evaluate the performance of the policy.

Scenario Type	**High Comm. Alg.**	**Low Comm. Alg.**
Rescue	ACBBA	CBAA
Search & rescue	ACBBA	CBAA
Fire monitoring	ACBBA	CBAA
Ship protection	PI	DHBA

Table 3.6: Algorithm specified by the policy for each level of communication and type of scenario.

3.6 Results

In Figs. 3.14, 3.15, 3.16 and 3.18, we show our compiled results for the two communication switching conditions tested and each type of scenario. We use trade-off analysis to demonstrate the difference in performance between the metareasoning policy and each individual algorithm used in the policy with respect to the metrics chosen: max distance traveled, max number of transmitted messages, and min distance to the ship (applicable only to the ship protection scenario).

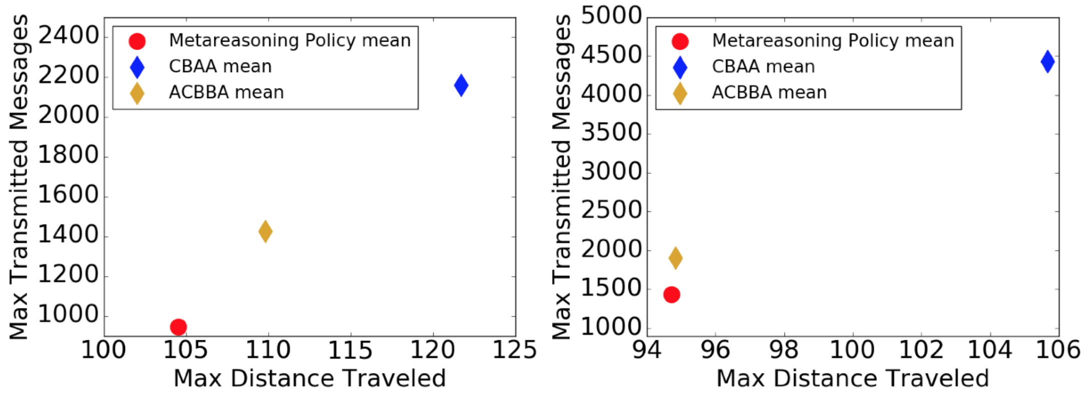


Figure 3.14: Trade-off analysis for the metareasoning policy and each individual algorithm used in the policy for the **rescue scenario**. High-low communication is shown on the left and low-high communication is shown on the right. Best expected performing methods are displayed as shaded circles. Methods with sub-optimal solutions are displayed as shaded diamonds.

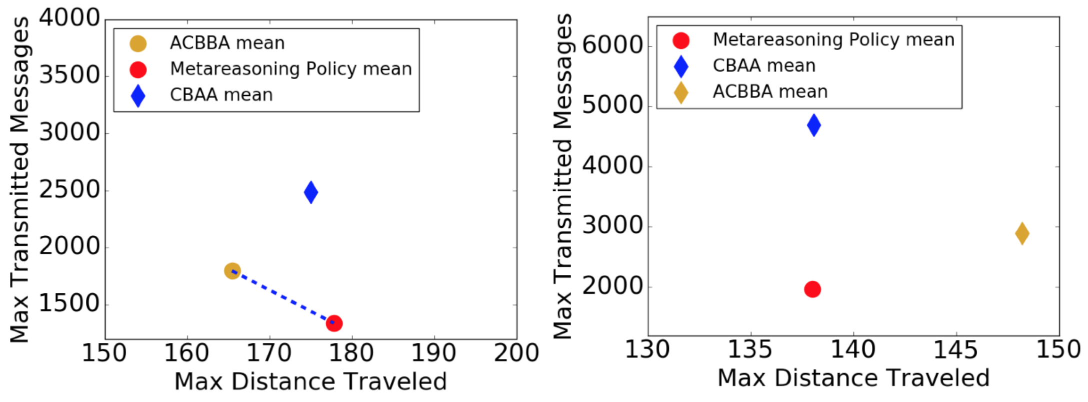


Figure 3.15: Trade-off analysis for the metareasoning policy and each individual algorithm used in the policy for the **search & rescue scenario**. High-low communication is shown on the left and low-high communication is shown on the right.

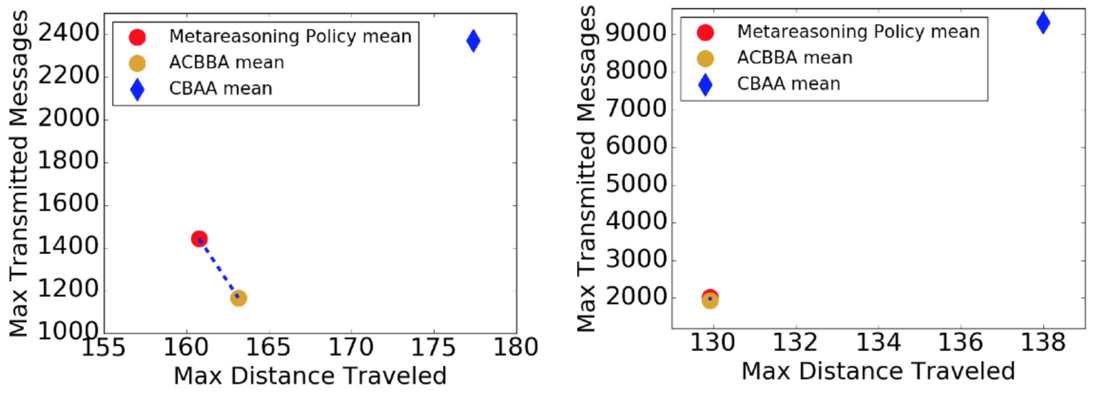


Figure 3.16: Trade-off analysis for the metareasoning policy and each individual algorithm used in the policy for the **fire monitoring scenario**. High-low communication is shown on the left and low-high communication is shown on the right.

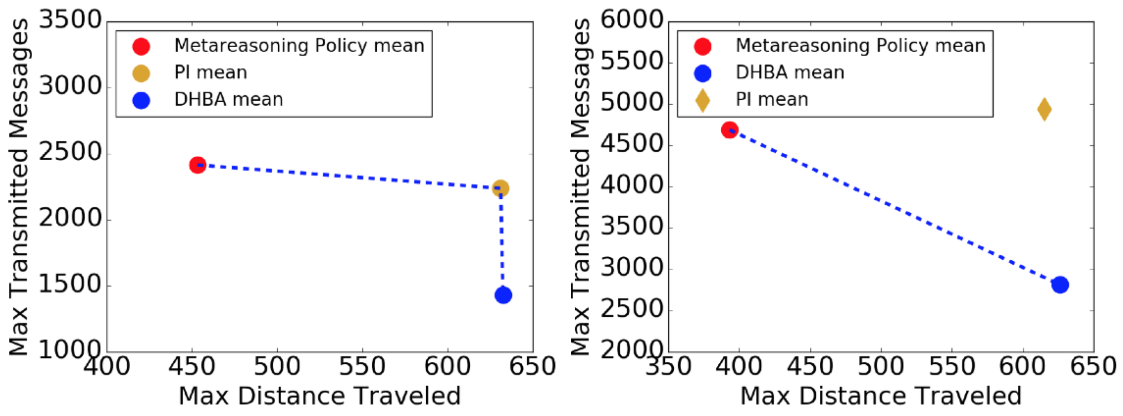


Figure 3.17: Trade-off analysis with respect to max distance traveled and messages transmitted for the metareasoning policy and each individual algorithm used in the policy for the **ship protection scenario**. High-low communication is shown on the left and low-high communication is shown on the right.

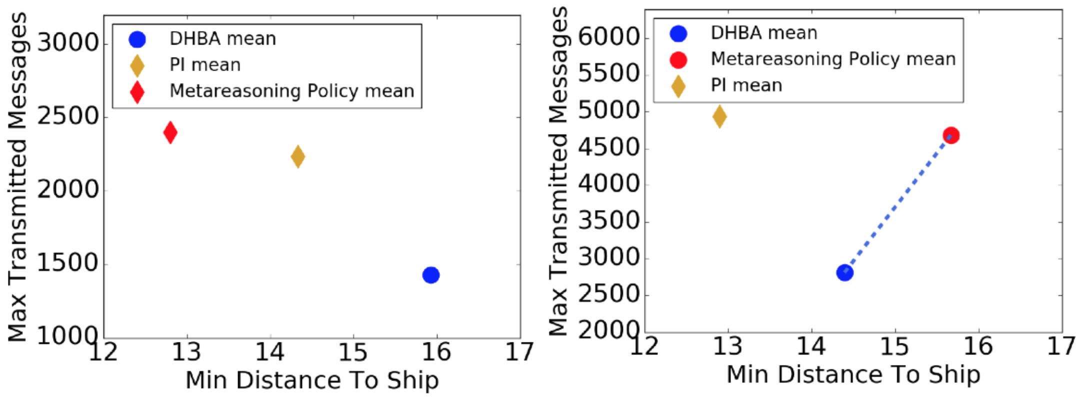


Figure 3.18: Trade-off analysis with respect to min distance to ship and messages transmitted for the metareasoning policy and each individual algorithm used in the policy for the **ship protection scenario**. High-low communication is shown on the left and low-high communication is shown on the right.

We make a few important observations from these results. First, for the rescue scenario, the metareasoning policy proposed was the single best expected performing strategy with respect to both performance metrics. For the search & rescue scenario, the metareasoning policy was the best expected performing strategy under low to high communication, however, it only performed better in terms of max number of transmitted messages under high to low communication. Consequently, we obtained a trade-off between ACBBA and the metareasoning policy.

In the fire monitoring scenario, the metareasoning policy provided a trade-off with ACBBA under high to low communication. In this communication scenario, the policy obtained the best expected performance in terms of max distance traveled while ACBBA resulted in a slightly lower number of transmitted messages. Under low to high communication, both ACBBA and the policy, are the non-dominated expected solutions. A possible reason for this is the target distribution, in which new fire targets appear near other targets. Agents can get an assignment of multiple nearby targets at once using ACBBA; thus, switching their allocation to a single-task assignment like CBAA may have spread out agents towards targets emerging in different fire clusters, resulting in a larger max distance traveled.

For the ship protection scenario, we observe that with respect to max distance traveled, the policy performed better on average compared to PI and DHBA. This occurred under both communication scenarios. With respect to min distance to ship, the metareasoning policy only performed better on average under low to high communication. Thus, the policy was more effective in terms of minimizing the max distance to track adversarial targets while DHBA was more effective in terms of maximizing the min distance to ship.

The results obtained demonstrate that on average, the policy outperforms running a single algorithm more consistently under low to high communication scenarios. Moreover, when agents experienced low communication at the start, metareasoning proved to be more effective when communication improved at a later time. However, when agents started with high communication, changing their task allocation scheme may not have been as effective. A reason for this may be that agents can reach consensus early on if they are able to communicate successfully at the beginning and may not benefit from changing their already computed solutions as communication degrades. It is important to note that across all communication scenarios, agents tend to perceive low communication as they initiate communication, however this is considered as a transient state of communication and is not explicitly included as part of the overall communication scenario.

Another observation we make is that the max number of transmitted messages for CBAA was consistently larger than the max number of transmitted messages obtained for the other two strategies. This can be explained by scenarios in which agents were not able to receive messages about completed tasks. For this reason, CBAA iterations continued to be executed until at least one agent successfully received updated information about the completion status of all tasks.

3.7 Conclusions and Future Work

In this chapter, we describe a metareasoning policy that a team of agents can execute to make effective meta-level control decisions based on the communication availability in the environment. Our policy was tested in various types of scenarios with different types of task allocation problems. We demonstrated that using the policy can lead to gains in performance or trade-offs in terms of max distance traveled and max number of messages transmitted compared to running a single fixed strategy. Using the LTL framework, this policy can be naturally extended to take into account additional environment features and desired reactive behaviors of the agents using LTL specifications. This would be an interesting direction for future work as well as exploring different metareasoning approaches that would not only execute a prescribed policy, but also modify the policy or generate new policies online.

In addition, with respect to communication, we would like to investigate the effects of more complex communication scenarios, for instance, scenarios in which only certain regions of the environment experience drops in communication quality. With respect to the metareasoning policy proposed, one possible extension is to include in the portfolio of task allocation algorithms considered, those that account for communication constraints and investigate the benefits of switching between such. This can also help us address questions on how to better utilize information exchanged among the agents and how to improve consensus under low communication.

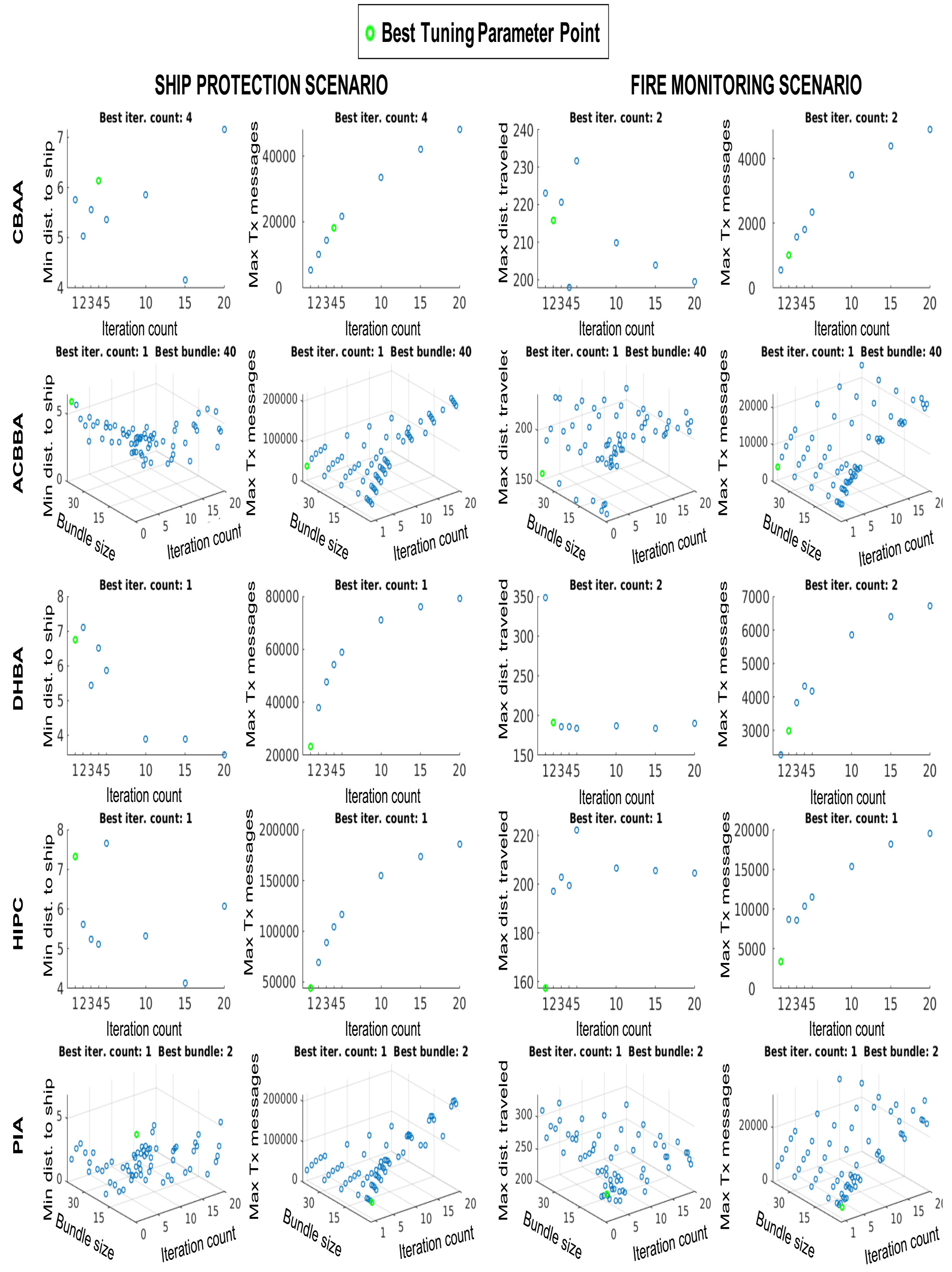


Figure 3.9: Tuning results for all 5 algorithms with respect to min distance to ship (ship protection scenario only), max distance traveled (fire monitoring scenario only) and max number of transmitted messages (both scenarios). The ship protection scenario is shown in the 1st and 2nd columns while the fire monitoring scenario is shown in the 3rd and 4th columns. Algorithms are shown in the following order: CBAA (1st row), DHBA (2nd row), HIPC (3rd row), ACBBA (4th row), PI (5th row).

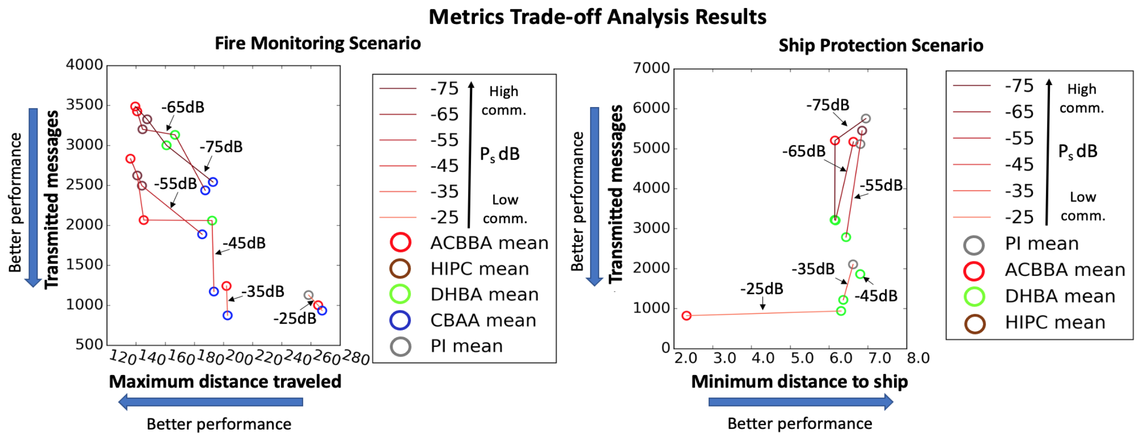
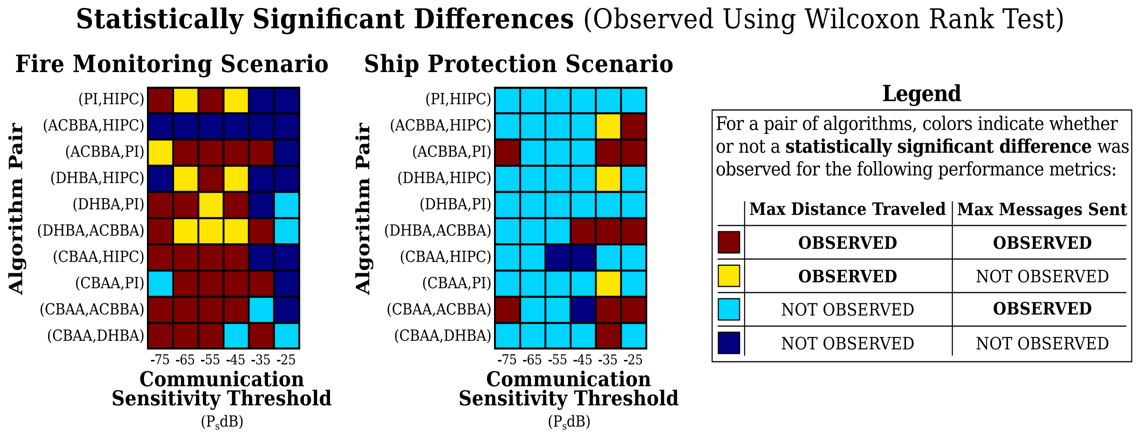


Figure 3.10: Metrics trade-off plot showing the best performing algorithms with respect to (max distance traveled or min distance to ship) and max number of transmitted messages for each level of communication tested.



For a pair of algorithms, colors indicate whether or not a **statistically significant difference** was observed for the following performance metrics:

	Max Distance Traveled	Max Messages Sent
🟥	**OBSERVED**	**OBSERVED**
🟨	**OBSERVED**	NOT OBSERVED
🟦	NOT OBSERVED	**OBSERVED**
⬛	NOT OBSERVED	NOT OBSERVED

Figure 3.11: For each communication level and pair of algorithms tested, we show the Wilcoxon Signed Rank test results for the fire monitoring scenario (left) and ship protection scenario (middle) with respect to the performance metrics evaluated.

Chapter 4: Object-Level Layer

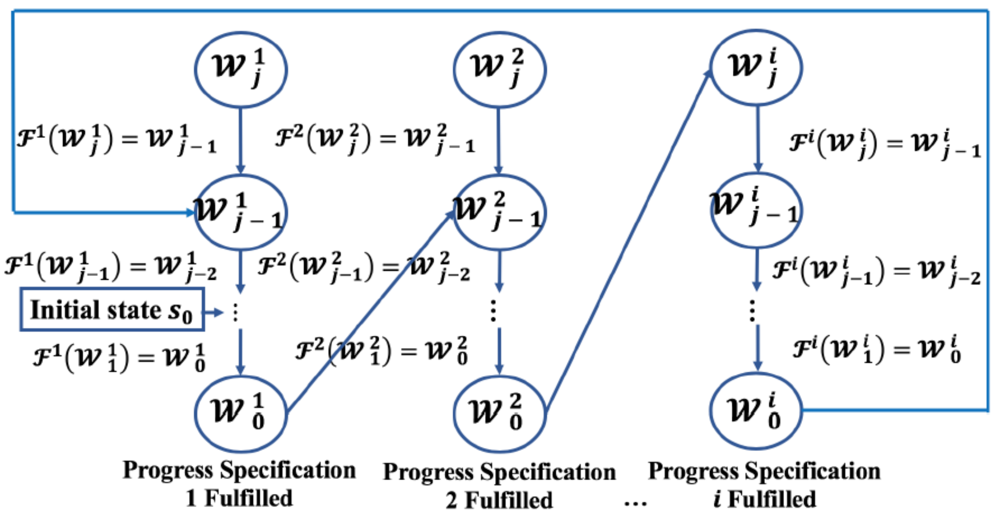
In this chapter, we describe the process of formulating reactive synthesis at the object-level layer of the agent. Our goal is to obtain a task and path planner for each agent in the team so that the team achieves a coordinated behavior. This coordinated behavior is accomplished by having each agent execute the synthesized high-level planner individually and employing dynamic allocation to assign different goals to each agent. Thus, we address state explosion, one of the main challenges of reactive synthesis, by employing a hierarchical integration of reactive synthesis, used to satisfy desired system design traits at the agent level, and dynamic allocation, used to coordinate the behavior of agents. However, a naive application of reactive synthesis to control a single agent can also lead to state explosion if dynamic environment settings are considered. We present a receding horizon framework to solve the synthesis problem efficiently and demonstrate the successful utilization of reactive synthesis for managing a team of UAVs in a firefighting application. We first begin by describing the receding horizon framework proposed.

4.1 Receding Horizon Framework

Previous works [3], [4], and [5] have explored reactive synthesis within a receding horizon (RH) framework. Within this framework, the state space for both the environment and system are partitioned into separate horizons, enabling the conversion of the given problem into a set of smaller problems for which to synthesize a controller. A single controller that satisfies the high-level system requirements and assumptions about the environment is then obtained by carefully combining the individually synthesized controllers.

The receding horizon implementation consists in segmenting the total system space into regions $\mathcal{W}_j$ for each progress specification. These regions are placed into properly constructed ordered sets $\mathcal{F}^i(\mathcal{W}_j^i)$ to ensure that the system variables will converge to satisfying each progress statement for the system progress goal at $\mathcal{W}_0$. Fig. 4.1 exemplifies the segmentation process of the total state space into an ordered set of defined regions $\mathcal{W}_j^i$, in which i denotes the system progress statement $i \in \mathcal{I}_g$, and j indexes the ordered regions $\mathcal{W}_j$ about i.

Each region $\mathcal{W}_j^i$ consists of its own GR(1) specification as shown in Eq. (4.1), and the goal of the synthesized controller is to drive the system states towards the next region within the ordered set so that the system eventually arrives at $\mathcal{W}_0$. For example, the controller will move states in region $\mathcal{W}_j^i$ towards region $\mathcal{W}_{j-1}^i$ in its attempt to fulfill progress specification i. Such specification is formalized as a GR(1)



$\mathcal{W}_j^1$
$\mathcal{F}^1(\mathcal{W}_j^1) = \mathcal{W}_{j-1}^1$
$\mathcal{W}_{j-1}^1$
$\mathcal{F}^1(\mathcal{W}_{j-1}^1) = \mathcal{W}_{j-2}^1$
Initial state s_0
$\mathcal{F}^1(\mathcal{W}_1^1) = \mathcal{W}_0^1$
$\mathcal{W}_0^1$
Progress Specification 1 Fulfilled
$\mathcal{W}_j^2$
$\mathcal{F}^2(\mathcal{W}_j^2) = \mathcal{W}_{j-1}^2$
$\mathcal{W}_{j-1}^2$
$\mathcal{F}^2(\mathcal{W}_{j-1}^2) = \mathcal{W}_{j-2}^2$
$\mathcal{F}^2(\mathcal{W}_1^2) = \mathcal{W}_0^2$
$\mathcal{W}_0^2$
Progress Specification 2 Fulfilled
...
$\mathcal{W}_j^i$
$\mathcal{F}^i(\mathcal{W}_j^i) = \mathcal{W}_{j-1}^i$
$\mathcal{W}_{j-1}^i$
$\mathcal{F}^i(\mathcal{W}_{j-1}^i) = \mathcal{W}_{j-2}^i$
$\mathcal{F}^i(\mathcal{W}_1^i) = \mathcal{W}_0^i$
$\mathcal{W}_0^i$
Progress Specification i Fulfilled

Figure 4.1: Segmentation process of the state space and example of ordered set flow-down performed in receding horizon framework [3].

specification as follows:

$$\begin{aligned}\Psi_j^i = ((s \in \mathcal{W}_j^i) \ \wedge \ \Phi \ \wedge \ \bigwedge_{i \in \mathcal{I}_r} \Box\varphi_{s,i}^e \ \wedge \ \bigwedge_{i \in \mathcal{I}_f} \Box\Diamond\varphi_{p,i}^e) \\ \longrightarrow (\bigwedge_{i \in \mathcal{I}_s} \Box\varphi_{s,i}^s \ \wedge \ \Box\Diamond(s \in \mathcal{F}^i(\mathcal{W}_j^i)) \ \wedge \ \Box\Phi).\end{aligned} \tag{4.1}$$

where s refers to the system state. The formula Φ represents all limitations on the system states in order to disallow system transitions to or initializing within invalid states. Thus, each synthesized controller prevents transitions to states that are infeasible for other horizons.

Before describing the implementation of the receding horizon framework in more detail, we first provide some motivation for the chosen application to firefighting and specific problem to be solved using this framework.

4.2 Application to Firefighting

We explore the design of a high-level mission planner and controller for managing unmanned aerial vehicles (UAVs) in a fire fighting application by having each agent execute a reactive controller and applying dynamic allocation of the UAVs as resources for fighting a growing fire.

The application of fighting wildfires presents various challenging opportunities for high-level reactive control design since it can involve an expansive objective space. Typical fire fighting scenarios involve numerous ground workers and piloted vehicles, including aerial vehicles capable of dropping large amounts of suppressant over regions of fire. Due to the chaotic nature of fires, numerous progress specifications tied to the environment dynamics can be generated to formally describe how a team of UAVs should slowdown the spread of fires. This serves as an apt example of the type of problem many reactive synthesis-based research endeavors have not addressed.

Managing wildfires is a relevant and complex challenge, straining the economy of the United States. In 2016, an amount exceeding \$63.5 billion was incurred due to damages caused by wildfires, and more than \$7.6 billion is spent annually to fight said fires [9]. As discussed in [10] and [11], UAVs pose a huge benefit to traditional firefighting methods, with the primary advantage of supplying additional "eyes in the sky". Previous work [8] has shown that aerial vehicles with suppression capabilities have been critical in slowing down the growth of fires due to their far greater range of maneuverability when compared to ground crews. Often these aircraft can change the outcome of a wildfire if used to attack a small fire early enough. Hardware demonstrations of UAVs extinguishing fires have been presented in [13], on a small, single-UAV scale, and through Lockheed Martin's use of the K-MAX and Stalker XE in [12]. Thus, there are many opportunities to leverage the use of mid-sized autonomous drones for quick response to fledging small-scale wildfires as well as limiting the number of required personale to a site.

Next, we present the problem formulation that exemplifies how reactive synthesis is used to construct a high-level controller, to be executed by each agent in a team of UAVs with the goal of slowing down the spread of multiple fires.

4.2.1 Problem Definition

We consider a 450-by-450 meter region of flat grassland, segmented by various large-scale obstacles. We assume fires spread from starting regions under fixed environmental conditions (e.g. wind speed and direction) and with any arbitrary initial conditions. A base of operations exists near the edge of the region and contains a fleet of N UAVs of moderately large size for fighting the fire.

Each UAV holds a varying level of suppressant for dumping on the fire, from High (100%), Medium (66%), Low (33%), to Empty (0%), associated with a total water volume of $W_v = 125$ liters. Each individual UAV contains a radio for communicating with base, GPS for determining position, and any other sensors required for lower-level controllers. Each UAV's average flight speed, v, is approximately 15 m/s. Design constraints on the UAVs require periodic landing and enforcement of in-flight kinematics resembling fixed-wing behavior. The design goal of the fleet is to significantly slow down the fires' spread to the outer edge of the domain as compared to the fires' natural growth. Fig. 4.2 visualizes the abstracted region for this problem.

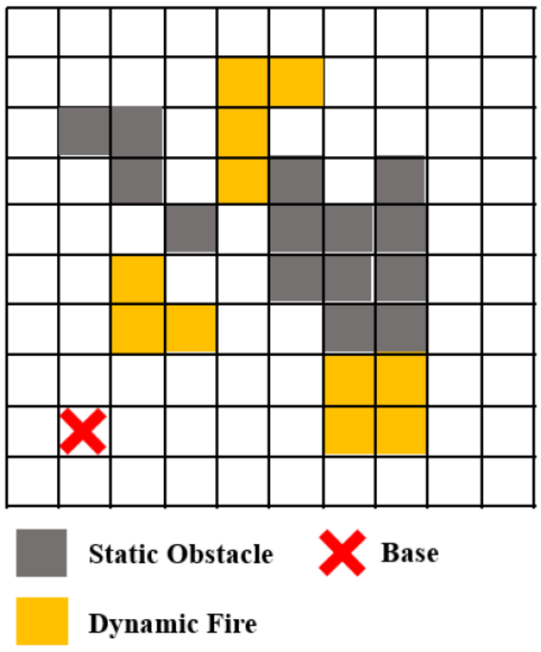


Figure 4.2: 2D grid partition of region with environmental indicators.

The formal definition of the abstracted system space is as follows.

Definition 1: The state set is defined as $S = S_p \times S_o \times W$, where the position set is $S_p = \{(1;1);(1;2);(2;1);...;(10;10)\}$, the orientation set is $S_o = \{0;90;180;270\}$, and the water level set is $W = \{0\%;33\%;66\%;100\%\}$. A single UAV at any given time is represented as an element $s \times S$. For the elements of s, $s_{x,y}$ is used to represent the position tuple, s_o is used to represent the orientation, and w is used to represent the water level. This implies that each position represents a cell of 45-by-45 meters. Furthermore, viable transitions for the UAVs between elements in the state space are defined assuming 3 transition scenarios: a sped up counterclockwise turn, a sped up clockwise turn, and a straight drive ahead. These transition scenarios result in the following transition system described in Definition 2, and visualized in Fig. 4.3.

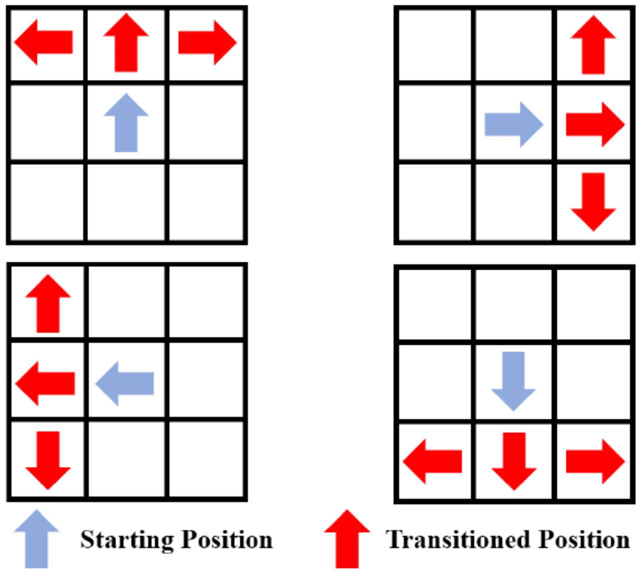


Figure 4.3: Possible transitions for UAV within grid given starting orientation and position.

Definition 2: The transition relation for the state set S is defined as $T = \{s \to s' \in R \subseteq S \times S\}$, where allowable transitions $s \to s' \in R$ are defined under the following conditions.

If $s_o = 0°$:

$$s' = \begin{cases} (s_x, s_y), & s'_o = 0° \\ (s_x + 1, s_y), & s'_o = 0° \\ (s_x + 1, s_y + 1), & s'_o = 90° \\ (s_x + 1, s_y - 1), & s'_o = 270° \end{cases} \tag{4.2}$$

If $s_o = 90°$:

$$s' = \begin{cases} (s_x, s_y), & s'_o = 90° \\ (s_x, s_y + 1), & s'_o = 90° \\ (s_x + 1, s_y + 1), & s'_o = 0° \\ (s_x - 1, s_y + 1), & s'_o = 180° \end{cases} \tag{4.3}$$

If $s_o = 180°$:

$$s' = \begin{cases} (s_x, s_y), & s'_o = 180° \\ (s_x - 1, s_y), & s'_o = 180° \\ (s_x - 1, s_y + 1), & s'_o = 90° \\ (s_x - 1, s_y - 1), & s'_o = 270° \end{cases} \tag{4.4}$$

If $s_o = 270°$:

$$s' = \begin{cases} (s_x, s_y), & s'_o = 270° \\ (s_x, s_y - 1), & s'_o = 270° \\ (s_x + 1, s_y - 1), & s'_o = 0° \\ (s_x - 1, s_y - 1), & s'_o = 180° \end{cases} \tag{4.5}$$

Depending on the location of static obstacles and boundaries, elements within the described transition relation should be avoided if they violate the reachability property of the system, as defined through Definition 3. This implies that the existence of obstacles requires limitation on allowable states S in the system specifications. A graph search for paths that lead to dead ends is an accessible way of determining these states.

Definition 3: The system S is reachable if there exists a path of states, composed of a finite number of consecutive transitions in $s \rightarrow s' \in T \subseteq S \times S$, such that all $s \in S$ can be eventually reached from any other initial state $s \in S$.

The formal definition of the abstracted environment space is provided in Definition 4. Note that this general environment definition is expansive in size (up to 2^{100+N} combinations, where N is the number of fires). Also note that this environment definition serves as an abstraction to the fire growth model described in more detail in Section 4.3.2.

Definition 4: The fire environment is defined as $F_{x,y} = (x, y) \times \{True, False\}$ for any valid choice of x and y in the system domain except for the base and obstacle locations. The element $f_{x,y} \in F_{x,y}$ corresponds to whether fire is present at (x, y) or not. The environment domain consists of the combination of all $F_{x,y}$ sets and the landing signal sets of each UAV, i.e. $E = F_{1,1} \times F_{1,2} \times F_{2,1} \times ... \times F_{10,10} \times S_{land,1} \times S_{land,2} \times ... \times S_{land,N}$, where $S_{land,i} \in \{True, False\}$ corresponds to the landing signal of the i-th UAV. Under these formal definitions on the system and environment, the desired design constraints for each UAV are as follows. First, each UAV must fly to any region in the state space associated with an active fire (Eq. (4.6)) and dump a fraction of its water supply if possible (Eq. (4.7)),

$$\phi_1^s = \bigwedge_{(x',y')} \Box(f_{x',y'} \longrightarrow \Diamond(s_{x,y} \leftrightarrow (x', y'))), \tag{4.6}$$

$$\phi_2^s = \bigwedge_{(x',y')} \Box((w > 0\% \ \wedge \ f_{x',y'} \ \wedge \ (s_{x,y} \leftrightarrow (x',y'))) \leftrightarrow \bigcirc w = w - 33\%). \tag{4.7}$$

Next, each UAV must return to base for replenishing water supplies when empty (Eq. (4.8)) and the water level must refill to the max level when the UAV reaches base (Eq. (4.9)). Outside of any condition that forces the water level to change, the water level must remain constant (Eq. (4.10)). Base is a proposition that is True when s_p matches the associated (x,y) tuple for the base,

$$\phi_3^s = \Box(w = 0\% \longrightarrow \Diamond base), \tag{4.8}$$

$$\phi_4^s = \Box((w = 0\% \ \wedge \ base) \leftrightarrow \bigcirc w = 100\%), \tag{4.9}$$

$$\phi_5^s = \Box((\neg \bigcirc w = 100\% \ \wedge \ \neg \bigcirc w = 0\%) \leftrightarrow \bigcirc w = w). \tag{4.10}$$

The UAVs may receive landing signals and must land in the next available region unaffected by fire (Eq. (4.11)),

$$\phi_6^s = \Box((s_{land,n} \ \wedge \ \neg f_{x,y}) \longrightarrow (\bigcirc s_{x,y} \leftrightarrow s_{x,y})). \tag{4.11}$$

Lastly, the order in which fires are addressed must be determined by their capability of reaching the domain edge, and UAVs must coordinate suppressing fires in a manner that increases their combined effect on the environment, a system specification represented by $\phi^s_{priority}$. This specification is an open area of design, one in which a formal description of the fire behavior model ϕ^e_{model} and system response $\phi^s_{priority}$ would require further environment and system definitions, specifically in relation to predicting and assessing the wildfire dynamics. In the reactive synthesis language, the problem we seek to solve is defined as the following total specification:

$$\phi^e_{model} \rightarrow \bigwedge_{i\in[1,6]} \phi^s_i \ \wedge \ \phi^s_{priority}. \tag{4.12}$$

Given the specifications ϕ^e_{model} and $\phi^s_{priority}$, our goal is to "synthesize" a controller that satisfies Eq. (4.12) through utilizing reactive synthesis efficiently while avoiding introducing further variables to the synthesis problem.

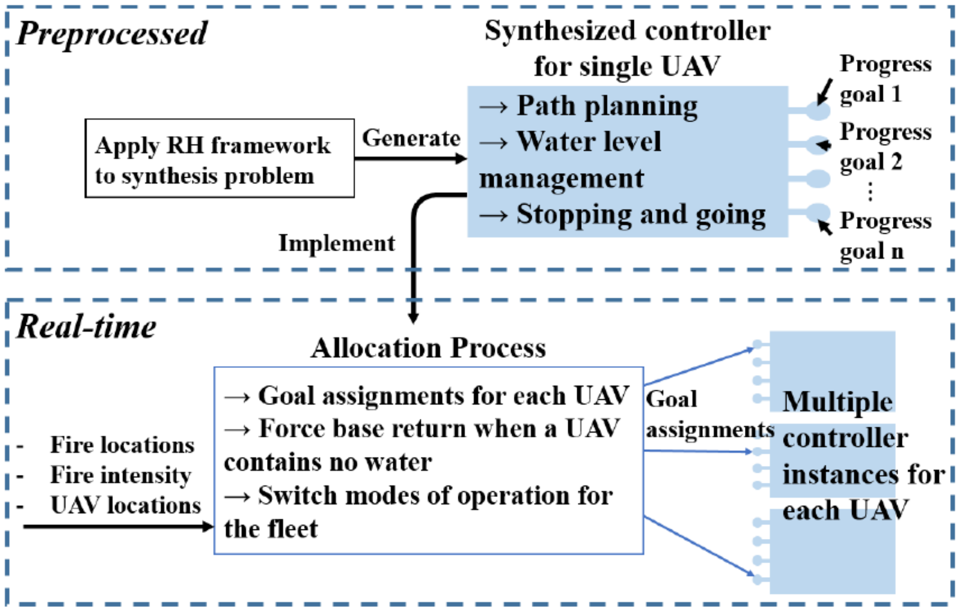
4.2.2 Proposed Solution

When the high-level design of a system does involve large scale environmental permutations and state spaces, solutions typically seek to discretize the synthesis problem. Discretization appears in the use of receding horizon control in [63] and the use of decentralized controllers for multiple agents in [15]. Both examples break down the top-level synthesis problem into smaller, discrete pieces for the computation benefits. On the other hand, [16] approached their synthesis problem with a focus on resolving deadlock under specific environment conditions instead of directly avoiding dynamic obstacles. In each of the presented cases, the problem description focused on a limited task space (i.e. the number of progress goals) and how the solution can handle larger sets of actions from an environment in relationship to safety specifications. The maximum state space sizes of [63]-[16] were up to the order of 100 variables (for [16] specifically), but the task space for each scenario explored never exceeded 4 progress statements. For the purposes of fighting growing, dynamic fires with a single autonomous UAV system represented by a state space abstraction size of 2 orders in magnitude or greater, a high-level controller created with just reactive synthesis could quickly present an impractical solution if the design has to accommodate a fairly granular environment space. Further expanding this concept to a whole fleet of UAVs, the problem worsens due to an increase in the number of system variables proportional to or greater than the number of UAVs, if considering centralized controllers. Even with decentralized controllers for each UAV, additional system variables might need to be introduced to describe coordination and behaviors between the decentralized UAV controllers.

To alleviate the computational complexity on reactive synthesis in this regard, we propose the coordination of UAVs be handled by a dynamic allocation process. Dynamic allocation presents an autonomous method for assigning resources in an ever-changing environment and determining which fires the UAVs should prioritize. For example, in [17], the resource allocation problem is framed as a multi-objective optimization problem of minimizing the extinguishing time and resource utilization cost, solved by the use of evolutionary algorithms. In [18], the fire behaviors and results due to resource allocation are formulated as a Markov Decision Process (MDP), from which a Monte Carlo tree search is used to determine the best areas to allocate resources. We implement a simplified optimization allocation strategy to address the assignments of UAVs for our fire fighting scenario. If the fleet of UAVs are treated as resources to manage with respect to a changing fire landscape, dynamic allocation would serve well in assigning the UAVs to specific fires. Assignments depend upon factors in the behavior of the fires such as density, ability to spread, wind and direction. An allocation algorithm is not necessarily constructed to control the UAVs within the state space defined, nor the algorithm manages other high-level system aspects associated with the UAVs, such as suppressant control and decisions on landings. Building these high-level behaviors into an allocation algorithm requires "handcrafting" these behaviors for all scenarios, an approach that synthesis, on the other hand, is well suited to avoid.

Related to our work, [19] first touched upon the idea of manipulating the receding horizon framework for decomposing the synthesized problem and decentralizing the planning procedure. For their purposes, this idea resulted in the ability of multiple agents to satisfy high-level specifications through only considering other agents that entered their local horizon. For our purposes, decentralizing the planning procedure allows for dynamic allocation to output the order of progress goals specified in the synthesized controller in real-time. Thus, our proposed solution integrates reactive synthesis within the receding horizon framework previously described and dynamic allocation. These two methods form a high-level planner and controller that fulfills the design constraints imposed on each UAV and dictates the behavior of each one as well as their collective maneuvers. Fig. 4.4 depicts a conceptual view of the process of creating our solution and the tasks that each method performs. Fig. 4.5 depicts the direct relationship between the allocation process and a synthesized controller.

As shown in Fig. 4.4, a common synthesized controller is created for each of the UAVs through the receding horizon framework with the duties presented. Given any arbitrary initial condition, the controller aims to progress to each viable parti-



Preprocessed
Synthesized controller for single UAV
→ Path planning
→ Water level management
→ Stopping and going
Progress goal 1
Progress goal 2
Progress goal n
Apply RH framework to synthesis problem
Generate
Implement
Real-time
Allocation Process
→ Goal assignments for each UAV
→ Force base return when a UAV contains no water
→ Switch modes of operation for the fleet
- Fire locations
- Fire intensity
- UAV locations
Goal assignments
Multiple controller instances for each UAV

Figure 4.4: Diagram of creating the solution method and the responsibilities and roles for both the synthesized controllers and the allocation process during real-time implementation.

tioned space. Each one of these progress goals is represented within Fig. 4.4 by the "nodes" protruding from each rectangle. The order that the controller meets these progress statements for a single UAV, shown previously through the ordering of W_i in Fig. 4.1, is not dictated by the synthesized controller as typically performed within the RH framework. Instead, the allocation process decides which progress specification any single UAV should pursue, represented within Fig. 4.5 by the switching of the "nodes"' order for any given moment in time, and the allocation process is responsible for ensuring that each "node" can and/or will be fulfilled. Hence, the allocation process prioritizes and assigns which goals a single controller should meet next in real-time. The combination of these two methods in the described manner works to highlight the strengths of each method. The synthesized controllers manage various system oriented aspects of the design and path planning

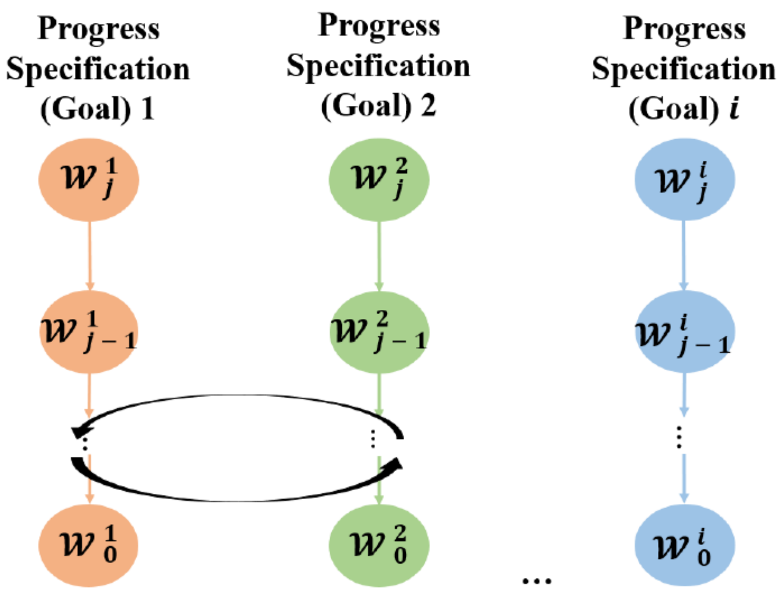

Progress Specification (Goal) 1
Progress Specification (Goal) 2
Progress Specification (Goal) i
w_j^1
w_{j-1}^1
w_0^1
w_j^2
w_{j-1}^2
w_0^2
w_j^i
w_{j-1}^i
w_0^i

Figure 4.5: Diagram of allocation process rearranging the progress goal ordering (as depicted in Fig. 4.1) for a single UAV controller instance in real-time.

while allocation governs the fleet behavior through assignments of goals for each controller.

Previously mentioned in [13], a physical scenario was constructed that dealt solely with one UAV gathering water, moving to another location, and dumping said water on the destination. Reference [13] demonstrates the existence of lower level controllers that could manage the individual actions necessary to achieve the high-level planner and controller proposed (at least for a smaller scale UAV). In Section 4.4, we describe an implementation for synthesizing low-level controllers satisfying reachability specifications in order to generate the required inputs to realize the discrete transitions presented in Fig. 4.3.

Next section describes the construction and operation of the synthesized controllers.

4.2.3 Synthesis of Controllers in the Receding Horizon Framework

The goal of the RH framework is to individually synthesize a controller about each progress goal while satisfying all safety specifications as formulated for the entire system. To reinterpret the specifications provided in Eqs. (4.6) and (4.8) in GR(1) form, we assume that the dynamic allocation process acquires the role of determining when the UAV should go to base or fire. For example, when the UAV's water level is empty, the dynamic allocation forces the UAV to prioritize the goal associated with base. Thus, the translated GR(1) goals involve always eventually driving the system to fire regions or the base, invoked by the presence of fire or absence of held water, respectively. As a result, Eqs. (4.6) and (4.8) are simply reinterpreted as Eqs. (4.13) and (4.14), respectively. Therefore, the specifications used in synthesis include the safety specifications shown in Eqs. (4.7), (4.9), and (4.11) alongside these simplified goal specifications,

$$\bigwedge_{x',y'} \Box\Diamond(s_{x,y} \leftrightarrow (x',y')), \tag{4.13}$$

$$\Box\Diamond base. \tag{4.14}$$

For the common synthesized controller, the formal environmental description described beforehand is broken down to the relative signals as perceived by a single UAV and changed by either the allocation method or the naturally occurring environment. These Boolean environment variables consist of: s_{land}, representing whether or not the UAV needs to stop due to a landing signal; f_d, representing the presence of fire directly beneath the UAV; and drop, representing the allocation process's signal to a UAV for dropping water on the assigned goal location. Hence, the relative environment $E_r = \{s_{land}, f_d, drop\}$. The environment specifications for a single UAV are listed as follows:

$$\phi^e_{init} = \{\}, \tag{4.15}$$

$$\phi^e_s = \{\}, \tag{4.16}$$

$$\phi^e_p = \Box\Diamond drop \ \wedge \ \Box\Diamond s_{land} \ \wedge \ \Box\Diamond\neg f_d. \tag{4.17}$$

We assume no guarantees on the environment's initial condition and safety behavior as stated in Eqs. (4.15) and (4.16). Eq. (4.17) states that always eventually allocation instructs the UAV to drop water, always eventually the UAV will experience engine failure, and always eventually the UAV will not fly over fire.

The system consists of S as described in Definition 1. Additional APs goal and base are created to indicate when the UAV enters a specified goal location and the base location, respectively. These revised definitions are used to form updated specifications in GR(1) form for each set of horizons about each goal. The system specifications are formally defined about each goal tied to a tuple (x, y) for each UAV:

$$\phi^s_{init} = \Psi. \tag{4.18}$$

Eq. (4.18) reflects that a UAV will start in a state allowed by Ψ (the RH tautology that governs feasible states). In this scenario, the tautology prevents the system from occupying any state that violates the conditions of Definition 3 before the horizons are applied. This disallows states in which obstacles are present or any states that only lead to obstacles.

$$\phi_{s,1}^{s} = \square((s_{land} \ \wedge \ \neg f_d) \leftrightarrow (\bigcirc s_{x,y} \leftrightarrow s_{x,y})). \tag{4.19}$$

Eq. (4.19) is the modified form of Eq. (4.11), already in GR(1) form, replacing the global $f_{x,y}$ element with the local f_d variable.

$$\phi_{s,2}^{s} = \square((w = 0\% \ \wedge \ goal \ \wedge \ base) \leftrightarrow (\bigcirc w = 100\%)), \tag{4.20}$$

$$\phi_{s,3}^{s} = \square((w > 0\% \ \wedge \ goal \ \wedge \ \neg base \ \wedge \ drop) \leftrightarrow (\bigcirc w = w - 33\%)), \tag{4.21}$$

$$\phi_{s,4}^{s} = \square((\neg \bigcirc w = 100\% \ \wedge \ \neg \bigcirc w = w - 33\%) \leftrightarrow (\bigcirc w = w)). \tag{4.22}$$

Eqs. (4.20), (4.21), and (4.22) are the modified versions of (4.9), (4.7), and (4.10), respectively. In Eq. (4.20), the system reaching the base location has been replaced with the local $goal \ \wedge \ base$ proposition combination to enforce the notion that UAVs only fill up when reaching base if the base was also set as the goal.

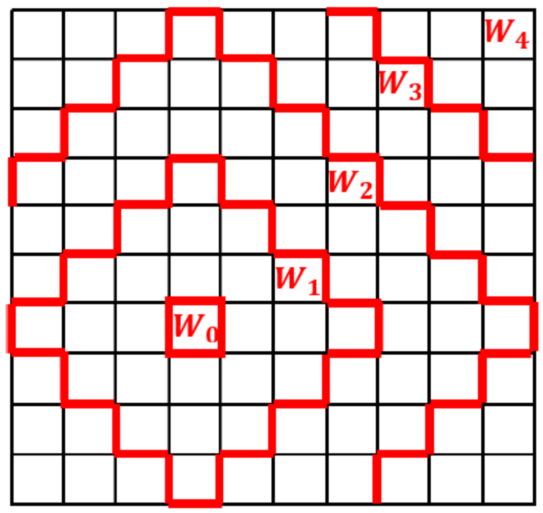
$$\phi_{p}^{s} = \square\Diamond goal. \tag{4.23}$$

Lastly, Eq. (4.23) is the generalized version of Eqs. (4.13) and (4.14). It models both since the allocation process is responsible for ensuring the conditions that drive the UAV to either the base location or a particular fire location, which serve as the current goal. Eqs. (4.18) - (4.23) along with the environment definitions form Eq. (4.24), the synthesis problem about each progress goal.

$$\Psi^{x',y'} = \phi^e_{init} \wedge \phi^e_s \wedge \phi^e_p \longrightarrow \phi^s_{init} \wedge \bigwedge_{i\in\{1,\ldots,4\}} \phi^s_i \wedge \phi^s_p \wedge \Phi. \tag{4.24}$$

This formulation in the RH framework captures the intended behavior of the original specifications while enabling the allocation method to dictate which goals are prioritized in which order.

To apply the RH framework, the state space must be segmented into horizons $\mathcal{W}_j^{x',y'}$ about every possible goal region. A formal definition for such is provided by Definition 5, and Fig. 4.6 visualizes the horizons for a single goal.



.

Figure 4.6: RH partitions for progress statement centered on position (4,4), i.e. $\mathcal{W}_0$

Definition 5: Horizons about (x', y') are defined as $\mathcal{W}_j^{x',y'} = \{s \in S \mid 3(j-1) \leq |s_x - x'| + |s_y - y'| \leq 3j, j = 1, 2, 3, \ldots\}$. $\mathcal{W}_0^{x',y'}$ corresponds to the goal.

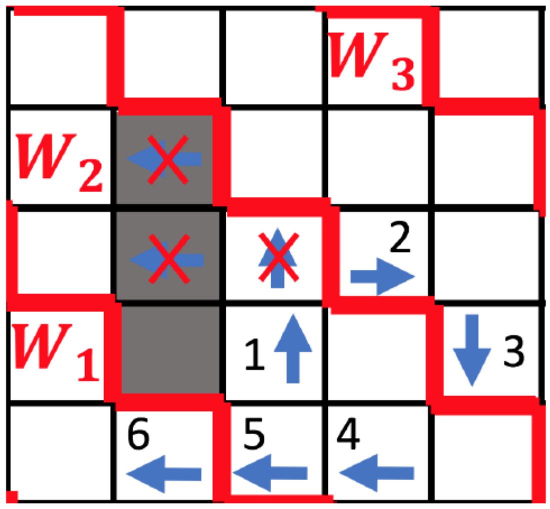


Figure 4.7: Example of an unrealizable state (1) in $\mathcal{W}_2$ for the horizon specification Ψ_2 that does have a valid next move (2) into $\mathcal{W}_3$ and subsequent path back to $\mathcal{W}_1$.

These horizons are utilized to form the individual $\Psi_j^{x',y'}$ specifications used in the RH framework. This formal definition is intuitive and easy to apply to this problem since it requires a simple calculation while automating the synthesis process. Unfortunately, these horizons do not account for static obstacle placements which create the restrictions represented by Φ, and blind application of the RH framework will create unrealizable specifications. A possible case of this issue is shown in Fig. 4.7, in which transitions from state (1) in $\mathcal{W}_2$ cannot go into $\mathcal{W}_1$ because of obstacles. Transitions that are allowed to "move back" to $\mathcal{W}_3$, however, can subsequently provide a path back to $\mathcal{W}_1$, as represented by the numbered arrows. However, this would violate the order along which the controller moves the system. To circumvent a horizon violation as such, a horizon modification algorithm is applied during synthesis to maintain the realizability of all RH specifications for the given environment with a static obstacle configuration, shown in Algorithm 1.

Algorithm 1: $\mathcal{W}_j^{x',y'}$ Modification During Synthesis

```
1: procedure SYNTHESIS_GOAL($x', y'$)    ▷ Synthesize controllers from all initial conditions for the $x', y'$ goal location
2:     for $0 \leq j \leq N$ do
3:         for $s \in \mathcal{W}_j^{x',y'}$ do
4:             Synthesize controller given $x', y'$ goal and current $s$
5:             if Controller == None then
6:                 Remove $s$ from $\mathcal{W}_j^{x',y'}$
7:                 Add $s$ to $\mathcal{W}_{j+1}^{x',y'}$
8:             end if
9:         end for
10:    end for
11: end procedure
```

4.2.4 Receding Horizon Modification And Proof of Specification Realizability

By applying Algorithm 1 during the synthesis and construction of controllers, we can verify the viability of all initial conditions. If no controller from a given initial condition is found that satisfies the specification, Algorithm 1 is used to automatically generate the needed modifications to the horizon template to render the specification realizable. Doing so during execution allows for keeping valid horizons initially provided and only changing them as needed for the given environment. In Theorem 1, we provide a proof of realizability for the specification under the modified horizon templates.

Theorem 1: Given a modified version of the system in Definition 1 which has restricted accessible states and transitions through the addition of static obstacles but still maintains the reachability property described in Definition 3, and by using the *initial horizons $\mathcal{W}_j^{x',y'}$ described in Definition 5 applied with the horizon modification algorithm described by Algorithm 1, the specification for each horizon surrounding each progress goal, $\Psi_j^{x',y'}$, will remain realizable, preserving the RH framework guarantees on the overall specification.*

Proof: First, we maintain that the overall specification (Eq. (4.24)) is realizable for the modified system description given no horizons and any allowable initial condition. Because of such, a horizon-based synthesized solution exists that fulfills the framework and definitions provided in [3].

Given the Definition 5 description of the receding horizons $\mathcal{W}_j^{x',y'}$ for any individual goal (x', y'), reachability (Definition 3) implies that for any state sequence that starts and leads from $s \in \mathcal{W}_j^{x',y'}$ to a state s_f with $s_{f,x,y} = (x', y')$, said sequence π must contain at least one $s \in \mathcal{W}_k^{x',y'}$ for all $0 \leq k \leq j$. Under the modified system definition, all available sequences for some $s \in \mathcal{W}_j^{x',y'}$ may also need to include $s \in \mathcal{W}_r^{x',y'}$ for some $r > j$, i.e. the only available path to the goal may require the state to move into horizons away from the goal before moving back through horizons towards the goal due to the presence of obstacles. The presence of such a sequence that includes paths with $s \in \mathcal{W}_{r>j}^{x',y'}$ as the only valid path immediately violates the order conditions for the receding horizon specification $\Psi_j^{x',y'}$. To address this violation, modifications to the horizons, as shown in Algorithm 1, are made during synthesis to maintain the condition that a path π leading from $s \in \mathcal{W}_j^{x',y'}$ does not contain any $s \in \mathcal{W}_{r>j}^{x',y'}$.

As controllers are synthesized around each goal and for each initial condition s_i within each set $\mathcal{W}_j^{x',y'}$, starting with $j = 0$ and incrementing, realizability failures are direct results of the lack of a system path to the horizon $\mathcal{W}_{j-1}^{x',y'}$ that remains only in $\mathcal{W}_{j}^{x',y'}$. This is a result of the failure to satisfy $\Psi_{j}^{x',y'}$ since the overall specification $\Psi^{x',y'}$ is realizable. Because a path starting at the initial condition s_i that fulfills the global specification must exist on the global scale and none of the sets $\mathcal{W}_{j}^{x',y'}$ overlap per index (x', y'), the path must enter into $\mathcal{W}_{j+1}^{x',y'}$ due to the reachability property stated before. Through the algorithm, this state s_i is removed from $\mathcal{W}_{j}^{x',y'}$ and added to $\mathcal{W}_{j+1}^{x',y'}$. All intermediate states between the initial condition and horizon $\mathcal{W}_{j+1}^{x',y'}$ are also moved to the next horizon since each state is tested as an initial condition in Algorithm 1, and these states cannot serve as viable initial conditions themselves. Therefore, the revised $\mathcal{W}_{j+1}^{x',y'}$ contains the original set $\mathcal{W}_{j+1}^{x',y'}$ plus all states from $\mathcal{W}_{j}^{x',y'}$ that could not serve as initial conditions to reach the next horizon of $\mathcal{W}_{j-1}^{x',y'}$ (or goal if $j = 0$). This statement serves as a recursive assignment for each horizon j, shifting states back horizons until a new horizon set for a goal is defined such that each $s \in \mathcal{W}_{j}^{x',y'}$ starts a path contained solely in $\mathcal{W}_{j}^{x',y'}$ that reaches $\mathcal{W}_{j-1}^{x',y'}$. Because of this, the receding horizon modification ensures that $\Psi^{x',y'}$ is realizable for all goals, all horizons, and all initial conditions, maintaining the guarantees provided by the RH framework used from [3].

Next section describes the allocation algorithm used for the high-level controller.

4.2.5 Dynamic Allocation

We apply a dynamic allocation process for assigning the UAVs to different fire perimeter locations that spread with time. This process considers four main attributes that directly affect a fire's progress to the boundary edge and a single UAV's ability to suppress such. These include: proximity of the UAVs to fire locations; proximity of fires to the domain boundary; wind direction and magnitude; and the amount of burn-through time provided by any suppressant acting on the fire. These attributes were chosen due to their simplicity in calculation and their ease of observation outside of the simulation environment. We assume that the fire model (which determines the previous attributes) and its corresponding perimeter is correctly abstracted into the cells used by the synthesis process (i.e. any cell that contains the fire perimeter is interpreted as holding fire and is made available for allocation). Algorithm 2 shows the process for allocating UAVs to the fire perimeter locations, performed at each update of the synthesized controllers. The inputs consist of UAVs, the set of all location tuples $s_{i,(x,y)}$ for the UAVs (ordered by the index i), and F_p, the set of all current fire perimeter location tuples f_p in the environment domain. Due to the limited number of abstractions that our synthesized controller acts over, calculating the cost of each individual fire relative to each UAV is a feasible option of determining the minimized allocation cost per UAV at each update of the synthesized controllers. Note that no optimization of UAV assignments is performed across members, i.e. the UAVs are assigned to fire locations in order and the next UAV cannot be assigned to a fire previously chosen in the main

Algorithm 2: UAV dynamic allocation process

```
1: procedure ASSIGN_UAVS($UAVs$, $F_p$)    ▷ Allocate all UAVs to fires in the fire perimeter set $F_p$
2:     for $s_{i,(x,y)} \in UAVs$ do
3:         $min\_cost \longleftarrow$ inf
4:         for $f_p \in F_p$ do
5:             cost $\longleftarrow g(f_p, s_{i,(x,y)})$
6:             if cost $\leq min\_cost$ then
7:                 $f\_min \longleftarrow f_p$
8:                 $min\_cost \longleftarrow$ cost
9:             end if
10:        end for
11:        Allocate $i^{th}$ UAV to $f\_min$
12:        Remove $f\_min$ from $F_p$
13:    end for
14: end procedure
```

loop. The cost function $g(f, x)$ is calculated as the weighted sum of: the distance between the fire and the UAV; the fire's distance from the closest boundary edge; the alignment of the fire's direction alongside the wind and its magnitude; and the amount of suppressant acting on the fire. This equation is displayed in Eq. (4.25),

$$\begin{aligned} g(f, x) = d_f * \parallel x - f \parallel + e_f * min(\parallel edge - f \parallel) \\ - w_f * \parallel f.wind \parallel + b_f * suppressant\_time\_left, \end{aligned} \tag{4.25}$$

where d_f, e_f, w_f, and b_f are heuristic weights for each relevant attribute. The coefficient d_f behaves as a penalty weight for the distance between a UAV location x and fire location f. The coefficient e_f represents a penalty weight on fires that are further from any location along the outer edge of the domain. The coefficient w_f is an importance weight on fires further along the direction of the wind. A greater value contributes negatively to the total cost. Finally, coefficient b_f corresponds to a penalty on the remaining time for suppressant already present at the fire. This increases a relative fire's cost proportionally to the remaining burn-through time. Within all four terms, the variable edge represents any location along the domain boundary, the vector wind corresponds to a scenario's x and y components of the wind vector, and the function *suppressant_time_left* calculates the remaining burn through time (in seconds) of suppressant at a current fire location, returning zero if no suppressant is present. These weights represent "knobs" for heuristically tuning the allocation of UAVs to individual fires. Ideally, we aim to have the UAVs prioritize fire perimeters that are moving further along the wind direction and approaching the edges of the domain foremost. The distance from fire and burn-through time terms act to "spread out" the allocation of UAVs when the wind and edge terms are less severe. Through initial testing, we achieved the desired behaviors using standard units by choosing the coefficients d_f, e_f, w_f, and b_f as 0.1, 1.0, 0.1, and 0.02, respectively. For perspective, the four attribute terms in the cost function without coefficient multiplication, using standard units when evaluated, were typically on the order of 100, 100, 10,000, and 1,000, respectively.

Next section describes the fire fighting scenarios considered and simulation used for testing our approach.

4.3 Implementation

To examine the effectiveness of the approach proposed in a fire fighting application, we first present the fire model used to generate test scenarios as follows.

4.3.1 Fire Simulation

A simplified fire simulation was constructed in Python following the wavelet differential equations and the Rothermel spread equation provided in FARSITE [21]. FARSITE models the fire fronts as propagating wavelets, with the fire spread rate calculated by the Rothermel spread equation serving as the main indication of intensity. Reference [22] provides the valuation of the Rothermel spread equation for various fuel types and climates, shown in Fig. 4.8. For the simulation, we assumed the fuel type was SH7, which is shrubbery in a dry climate, and approximated the spread rates for 3 levels of wind speed. Table 4.1 provides these approximations for each wind speed.

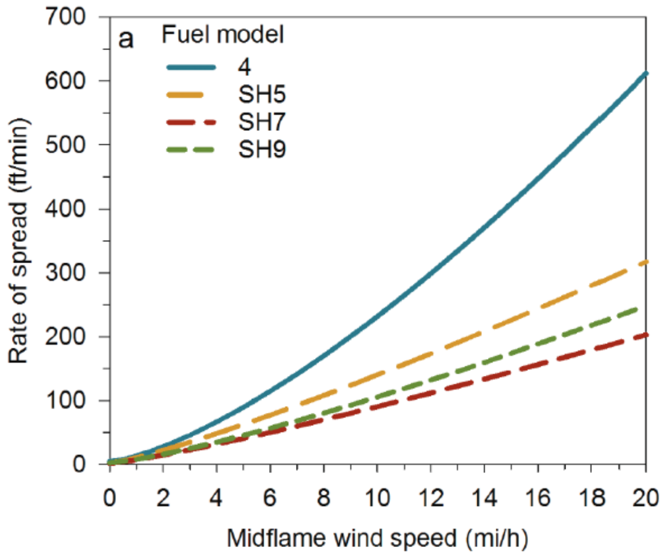


Figure 4.8: Fire spread rates as a function of wind speed for various fuel models, pulled directly from [22].

At each update time for the fire, the calculated spread rate of a fire vertex along the perimeter was also adjusted with a standard deviation to provide further

Wind Speed (m/s)	Rate of Speed (m/min)
1.0 (Low)	3.0
4.0 (Medium)	18.0
8.0 (High)	48.0

Table 4.1: Chosen wind speeds and resulting spread rates of fires for use in simulation scenarios.

variation in the fire front growth across each simulation. Additionally, the fire front model's perimeter was abstracted into distinct fire regions for use by the dynamic allocation process. Fire suppression mechanics are a relatively unknown area of research, and simulations typically assume that obstacles (both static and dropped suppressant) simply stop or greatly slow the spread rate at that specific location on a fire front, either indefinitely for static obstacles or a limited time for suppressant (referred to as the burn-through time). The placement of temporary obstacles in our simulation provided the direct mechanisms in which the fire was slowed down at any point by a UAV. Note that we only modeled slowing the fire down or stopping, no permanent extinguishing of cells.

The dropping of suppressant was modeled after the line length and burn-through times for 1500 liters of suppressant, as discussed in [24]. For a 1500 liter suppressant drop across 45 meters, the burn-through time is approximately 2 hours. We assumed that a linear relationship exists between the amount of suppressant and the burn-through time, i.e. 125 liters constitute about 10 minutes of burn-through, and each fractional drop of 125 liters on an area added the same proportional amount of 10 minutes to the current burn-through time. Additionally, we assumed a linear relationship between the rate of growth of a fire vertex within a suppressant area and the burn-through time left. The endpoints on this linear relationship were 0%

of normal growth rate for full burn-through time left and 5% of normal growth for no burn-through time left. This relationship was included to add conservative stress on the UAVs' ability to suppress the fires. Lastly, the geometry associated with suppressant drops was ignored, i.e. any suppressant dropped on the fire front was assumed to align itself so as to correctly block the fire within that region. The UAVs were modeled to follow simplified kinematics as described in Section 4.4 and a low-level controller was constructed and executed to maintain the transitions described in the problem description at each time step. Additionally, the UAVs were assumed to require 4 minutes anytime they were forced to stop, either by a random stop signal or stopping at the base to pick up suppressant. Various fire scenarios' initial conditions and parameter settings were constructed for the purposes of testing the controllers. These fire scenarios were aimed at testing the capabilities of the UAVs to slow down all fires from reaching the borders and gauge the effectiveness of different fleet numbers. These scenarios are provided in Fig. 4.9. The simulation cycled through multiple iterations on each scenario, testing the effectiveness of up to 4 fleet members. The average time for the fire to reach the outer edge on each scenario and fleet number combination was calculated. For all scenarios, the UAVs started at the base location.

4.3.2 Results

TuLiP [23] was utilized to realize and synthesize the controllers associated with each region $\mathcal{W}_j^{x',y'}$. On an Intel i5-6500 CPU @ 3.20 GHz processor, this total

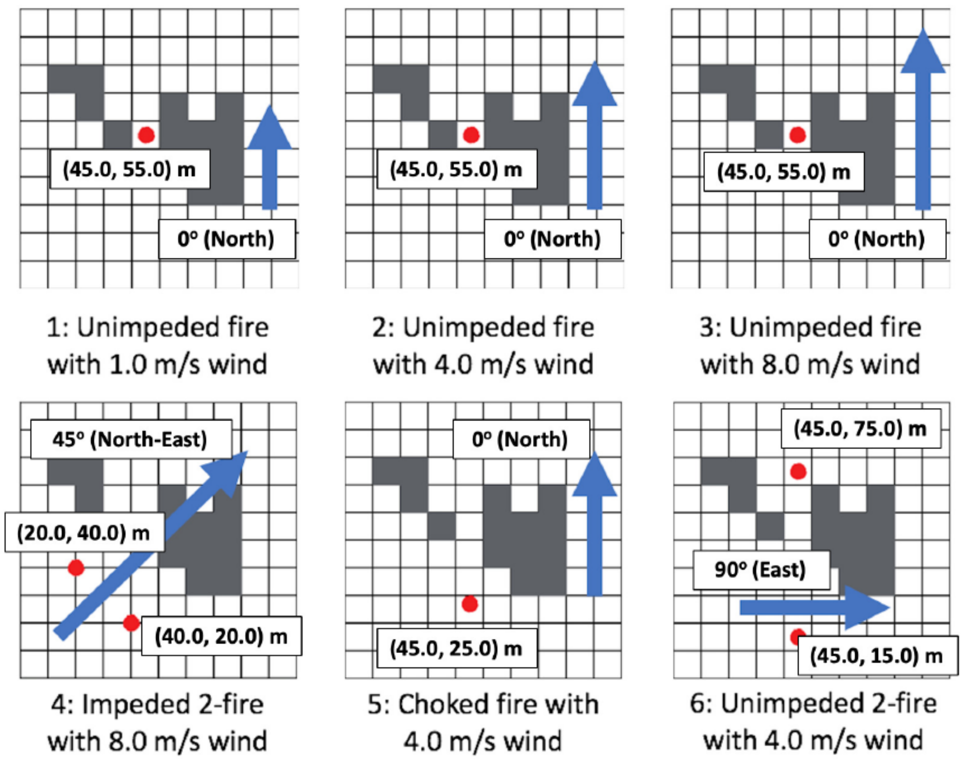

(45.0, 55.0) m
0° (North)
1: Unimpeded fire with 1.0 m/s wind
(45.0, 55.0) m
0° (North)
2: Unimpeded fire with 4.0 m/s wind
(45.0, 55.0) m
0° (North)
3: Unimpeded fire with 8.0 m/s wind
45° (North-East)
(20.0, 40.0) m
(40.0, 20.0) m
4: Impeded 2-fire with 8.0 m/s wind
0° (North)
(45.0, 25.0) m
5: Choked fire with 4.0 m/s wind
(45.0, 75.0) m
90° (East)
(45.0, 15.0) m
6: Unimpeded 2-fire with 4.0 m/s wind

Figure 4.9: Initial conditions of all fire scenarios, showing locations of 25 m diameter starting fires (red dots) and constant wind conditions (blue arrows). Grey boxes represent static obstacles.

process, approximately 250 regions $\mathcal{W}$, took on the order of 8 hours. In addition to the large amount of time to synthesize all of the individual controllers, numerous memory issues came up throughout the process, even with a system limit of 16 GB of RAM. The total size of the synthesized controllers was approximately 2 GB. For each scenario tested, simulations where conducted 100 times to assess the fleet of UAVs' ability to slow down the spread of the fire to the domain edges, provided each UAV experiences a 1% chance of a random stop signal for every transition time (3 seconds in all scenarios).

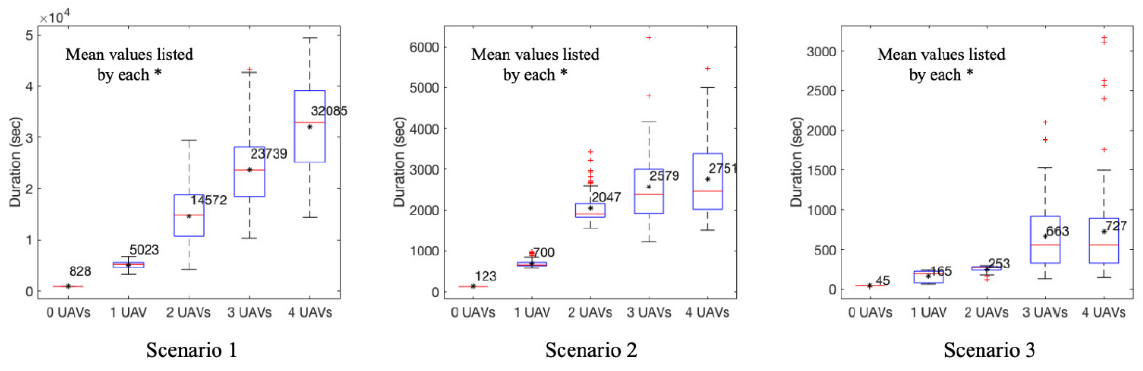


Figure 4.10: Box plot distributions of the simulated time for fire to reach any domain edge given a set number of UAVs fighting such fires (scenarios 1, 2, and 3).

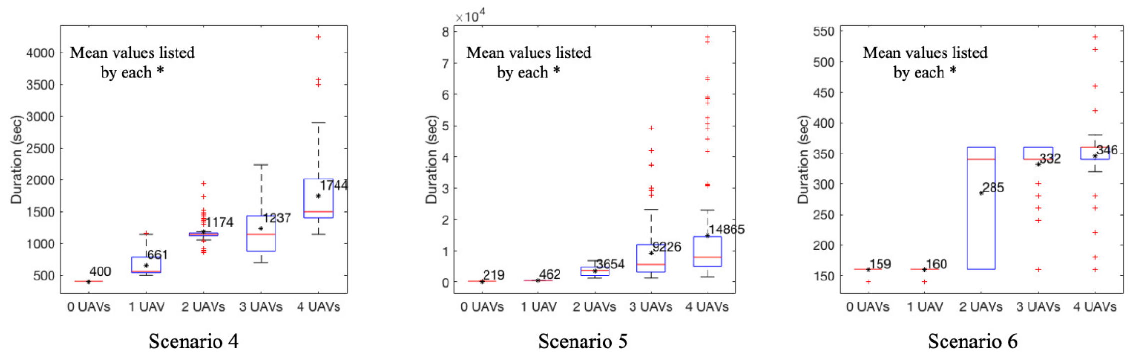


Figure 4.11: Box plot distributions of the simulated time for fire to reach any domain edge given a set number of UAVs fighting such fires (scenarios 4, 5, and 6).

The results were compiled and displayed in box plots shown in Fig. 4.10 and 4.11. The top and bottom of the boxes indicate the 75^{th} and 25^{th} percentiles, respectively, with red plus signs showing outliers. Median duration times of a distribution are represented by the middle red lines in the boxes, and mean duration times are shown as the asterisks alongside numerical values.

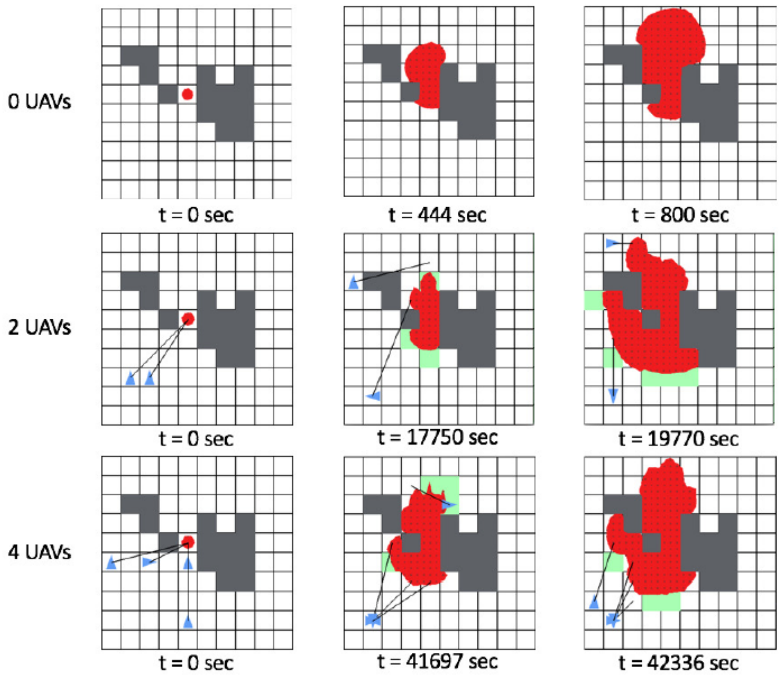


Figure 4.12: Box plot distributions of the simulated time for fire to reach any domain edge given a set number of UAVs fighting such fires (scenarios 4, 5, and 6).

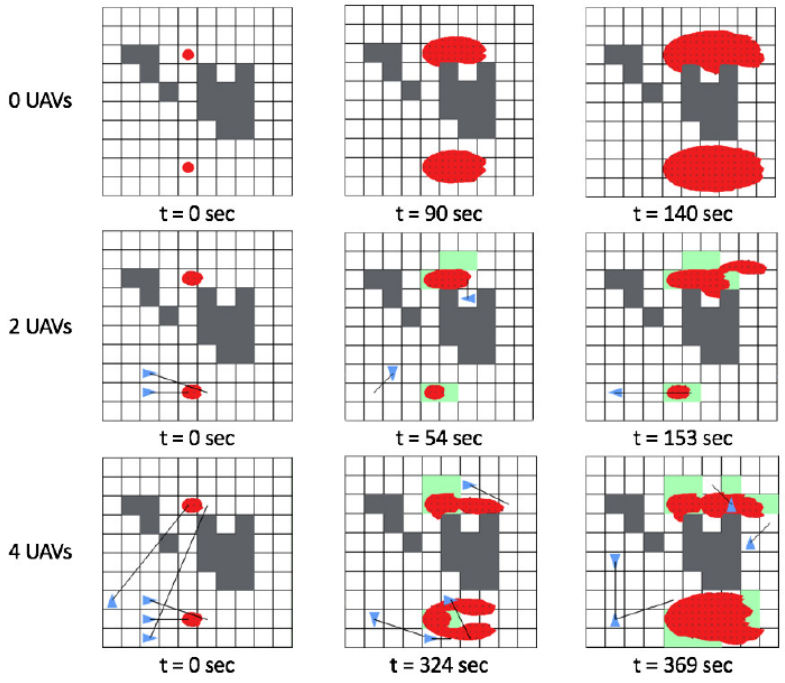


Figure 4.13: Box plot distributions of the simulated time for fire to reach any domain edge given a set number of UAVs fighting such fires (scenarios 4, 5, and 6).

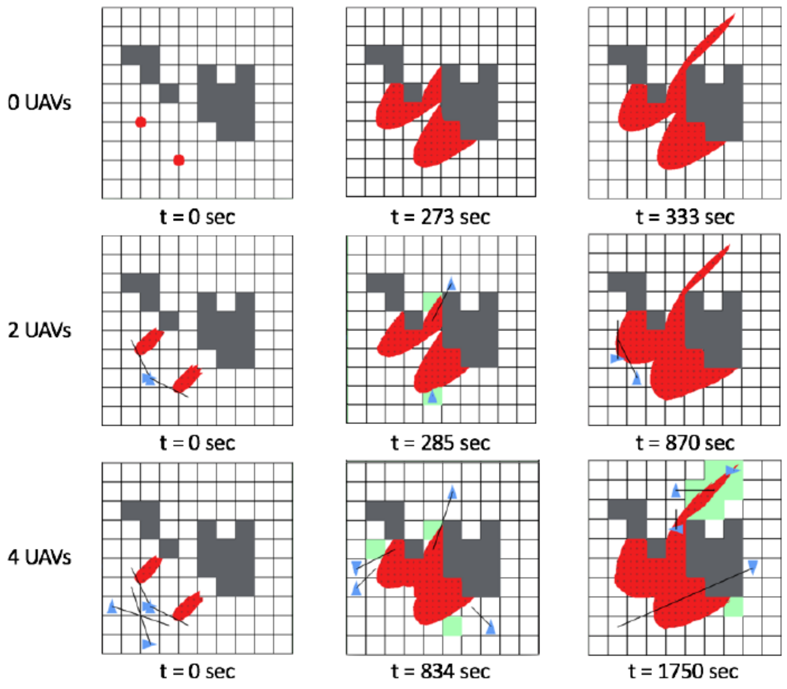


Figure 4.14: Box plot distributions of the simulated time for fire to reach any domain edge given a set number of UAVs fighting such fires (scenarios 4, 5, and 6).

Additionally, example time lapses for Scenarios 1, 4, and 6 are presented in Figs. 4.12, 4.13 and 4.14, showcasing how the fire grew in response to a varying number of UAVs. A few outcomes are observable through Figs. 4.10 and 4.11. First, in all cases, increasing the number of UAVs generally increased the median and average duration times of the tests, an intuitive result. Additionally though, especially evident in the Scenario 1 side of Fig. 4.10, the greater the number of UAVs used resulted in a higher spread between the minimum and maximum test duration times. The most likely explanation for this behavior is that greater differences between fire conditions in separate simulations accumulate over longer run-times associated with larger UAV groups, and since the UAVs are suspect to 4 minute periods of stopping while the standard 33% of 125 liters suppressant dropped corresponds to 3.33 minutes of burn-through time, these differences can greatly effect the UAVs' ability to slow down critical fires in time before refilling. The greater stress scenarios in 4 and 6 (greater wind and number of fires) provide notable results in contrast to one another. In scenario 4, the obstacles provide additional blockage for the UAVs, and as a result, a greater number of UAVs provides greater performance since the number of critical fire locations (e.g. fires further in the wind direction and closer to the edge) are limited and easily accessible in time. This is evident in Fig. 4.13 in the 285 second time of the 2 UAV case. By only hitting the edges of the fire about to wrap around the obstacle, the fire was greatly slowed down. On the other hand, in scenario 6, few obstacles slowed the fires down, and the UAVs had to "rush" in time to suppress the fires. Multiple UAVs were always required for the test to have any chance of lasting longer than the 0 UAV case, but often UAVs could not reach the critical fires in time, evident in both the 2 UAVs and 4 UAVs cases of Fig. 4.14 and by the minimum duration values in the scenario 6 side of Fig. 4.11.

4.4 Low-Level Controller

Provided the transitions in Fig. 4.3, we now describe an implementation of a low-level controller that enables the system to execute these transitions subject to the vehicle dynamics and some disturbance assumptions. We perform Hamilton-Jacobi (HJ) reachability analysis to compute the reach-avoid set corresponding to each transition and the optimal control that drives the system to the target set from its initial position, thus enabling the system to execute the given transition. We describe the HJ reachability analysis process in the following section.

4.4.1 HJ Reachability Analysis

HJ reachability analysis is a formal verification method for guaranteeing performance and safety properties of dynamical systems [25, 26, 27, 28]. We choose this method to synthesize the low-level controller due to its compatibility with general nonlinear system dynamics, formal treatment of bounded disturbance and control, and its ability to represent sets of arbitrary shapes [29]. In reachability analysis, the goal is to compute the reach-avoid set, defined as the set of states from which the system can be driven to a target set while satisfying time-varying state constraints at all times (see Fig. 4.15). One drawback of this method, however, is the cost of computational complexity for highly-dimensional systems (> 6 dimensions) [29]. For simplicity and efficient computation, we model the UAV as a 4D system, described in the next section.

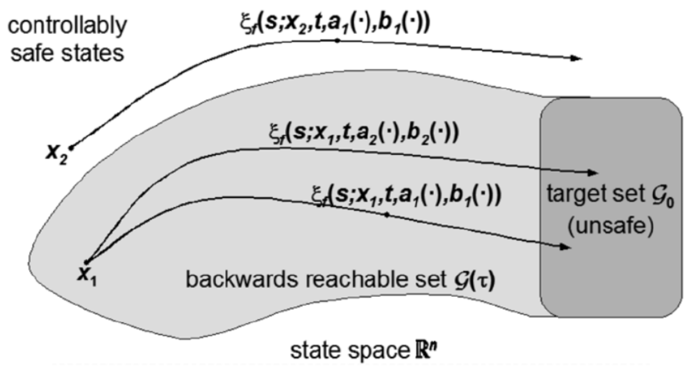

controllably safe states
$\xi_f(s;x_2,t,a_1(\cdot),b_1(\cdot))$
x_2
$\xi_f(s;x_1,t,a_2(\cdot),b_2(\cdot))$
$\xi_f(s;x_1,t,a_1(\cdot),b_1(\cdot))$
target set $\mathcal{G}_0$ (unsafe)
x_1
backwards reachable set $\mathcal{G}(\tau)$
state space $\mathbb{R}^n$

Figure 4.15: Example of target set and backward reachable set. Input signal $a(.)$ is chosen to drive the trajectory away from the target set, while input signal $b(.)$ is chosen to drive the trajectory toward the target. Figure taken from [30].

Consider a system state $x \in \mathbb{R}^n$, which evolves according to the ordinary differential equation (ODE)

$$\dot{x} = f(x(t), a(t), b(t)), t \in [t_f, 0], a(t) \in \mathcal{A}, b(t) \in \mathcal{B}, \tag{4.26}$$

where $a(t)$ and $b(t)$ denote the input of Player 1 and Player 2 respectively. The system dynamics, $f : \mathbb{R}^n \times \mathcal{A} \times \mathcal{B} \longrightarrow \mathbb{R}^n$ is assumed to be uniformly continuous, bounded, and Lipschitz continuous in x, while $a(\cdot) \in \mathbb{A}$ and $b(\cdot) \in \mathbb{B}$ are assumed to be measurable functions. For robust control problems, which apply non-anticipative strategies, one wants to obtain the robust control (Player 1) with respect to the worst-case disturbance (Player 2), which can then be modeled as an adversary with the instantaneous informational advantage. This advantage allows Player 2 to factor in Player 1's choice of input and adapting its own input accordingly for the purposes of establishing safety and reachability guarantees under the worst-case scenarios. That is, for a given control, we want to consider the worst disturbance out of all possible disturbances so that the system chooses its next control action to drive it back to its desired trajectory.

For our implementation, we can think of Player 1 as trying to steer the UAV towards the target, and Player 2 as trying to steer the UAV away from the target. Thus, we need to find the set of states such that the trajectories that start from this set can reach some given target set. Consequently, we compute the backward reachable set (BRS) of the dynamical system defined as

$$\mathcal{G}(t) = \{x : \exists \in \Gamma(t), \forall b(\cdot) \in \mathbb{B}, \zeta(0; x, t, \gamma[b](\cdot), b(\cdot) \in \mathcal{G}_0\}, \tag{4.27}$$

where $\Gamma(\cdot)$ in Eq. 4.27 denotes the feasible set of strategies for Player 1. The computation of the BRS requires solving a differential game between Player 1 and Player 2, which is then transformed into a differential game of degree using level set methods. Formally, the goal is to find a Lipschitz function $g(x)$ such that $\mathcal{G}_0$ (the target set) is equal to the zero sublevel set of g, that is, $x \in \mathcal{G}_0 \Leftrightarrow g(x) \leq 0$. Let J be the cost function defined as

$$J_t(x, a(\cdot), b(\cdot)) = g(x(0)), \tag{4.28}$$

then the system reaches the target set under controls a and b if and only if

$$J_t(x, a(.), b(.)) \leq 0. \tag{4.29}$$

Since Player 1 wants to drive the system to the target set, its control input seeks to minimize J, while Player 2 seeks to maximize it. Thus, the BRS is obtained as

$$\mathcal{G}(t) = \{x : G(t, x) \leq 0\}, \tag{4.30}$$

where $G(t, x)$ satisfies the following HJI PDE:

$$\begin{aligned} D_t G(t, x) + H(t, x, \lambda) &= 0, \\ G(0, x) &= g(x). \end{aligned} \tag{4.31}$$

The Hamiltonian is given by

$$H(t, x, \lambda) = \min_{a \in \mathcal{A}} \max_{b \in \mathcal{B}} \lambda \cdot f(x, a, b). \tag{4.32}$$

Finally, the optimal control for Player 1 to reach the target set despite the worst-case disturbance is obtained as

$$a^*(t, x) = \arg \min_{a \in \mathcal{A}} \max_{b \in \mathcal{B}} \lambda \cdot f(x, a, b). \tag{4.33}$$

4.4.2 UAV Continuous Dynamics

To simplify computation complexity, we model the UAV as a 4D system with the following dynamics:

$$\begin{aligned} \dot{x} &= v \cos\theta + d_x \\ \dot{y} &= v \cos\theta + d_y \\ \dot{\theta} &= w \\ \dot{v} &= a \end{aligned} \tag{4.34}$$

where (x, y, θ) represent the pose (position and heading) of the 4D UAV model, and v is the speed. The control of the 4D model consists of the linear acceleration, a, and the angular velocity, w. The disturbances are (d_x, d_y). The model parameters are chosen to be $a \in [-0.5, 0.5]$, $|d_x|$, $|d_y| \leq 0.5$, and $|w| \leq 0.2$.

4.4.3 Results

We implement reachability analysis using the level set toolbox *toolboxLS* and the Hamilton-Jacobi optimal control toolbox *helperOC* available in MATLAB. The reach and avoid sets are defined for each of the three possible transitions generated by the high-level controller and for each orientation. For illustration, let's assume the initial orientation for the UAV is 0° and the UAV is located in a square region defined by $\mathcal{C} = \{(x, y, \theta) : x_{lb} \leq x \leq x_{ub}, y_{lb} \leq y \leq y_{ub}\}$, where x_{lb} and y_{lb} are the lower bounds in the x and y direction respectively, while x_{ub} and y_{ub} are the upper bounds of $\mathcal{C}$. Note that since no inequality is defined for θ, all possible values for θ are contained in $\mathcal{C}$. Let $\mathcal{R}$ be a region defined as $\mathcal{R} = \{(x, y, \theta) : x_{lb} \leq x \leq x_{ub} + 2\delta, y_{lb} - \delta \leq y \leq y_{ub} + \delta\}$, where $\delta = x_{ub} - x_{lb} = y_{ub} - y_{lb}$. Assuming the UAV is directed to perform a straight drive ahead transition, we set the target set as $\mathcal{T} = \{(x, y, \theta) : x_{ub} \leq x \leq x_{ub} + \delta, y_{lb} \leq y \leq y_{ub}\}$. We then perform set exclusion on $\mathcal{R}$ to define the avoid set $\mathcal{A} = \mathcal{R} \setminus \mathcal{T}$. This definition of reach and avoid sets ensures that the UAV moves only within the appropriate discretized regions induced by the high level controller actions. Fig. 4.16 shows reach and avoid sets for a UAV with pose $(35, 45, 0)$, and all possible transitions. A similar process is performed to define the reach and avoid sets for each combination of transition and orientation. The discretization step for time is set to 0.5s.

Fig. 4.17 shows the resulting backward reachable set for the straight drive ahead (forward) transition, which reaches the initial position at time $t = 26$s. Fig. 4.18 shows the trajectory of the UAV after performing a straight drive ahead transition followed by a a sped up counterclockwise turn, then followed by a sped up clockwise turn transition. Figs. 4.19 and 4.20 show the optimal control for acceleration and angular velocity, respectively, obtained by the reachability analysis method.

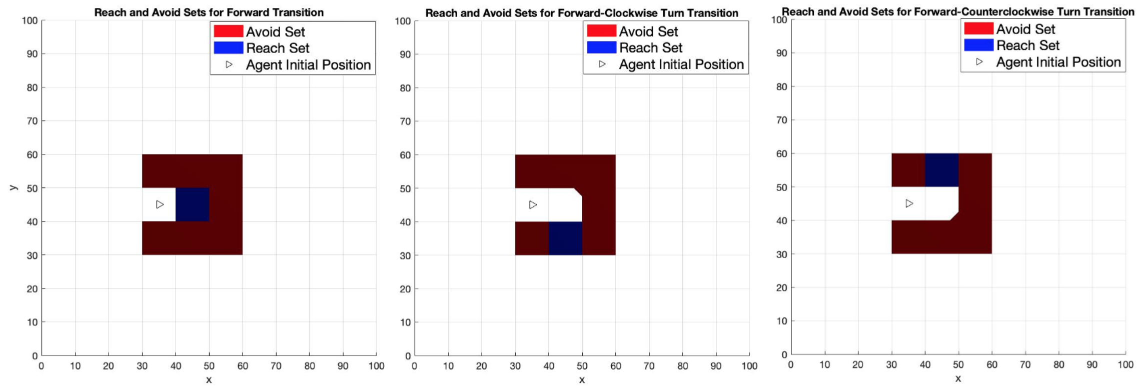


Figure 4.16: Reach and avoid sets defined for the forward transition (left), the forward and clockwise turn transition (middle), and forward and counter-clockwise turn transition (right). Initial pose of UAV is $(35, 45, 0)$.

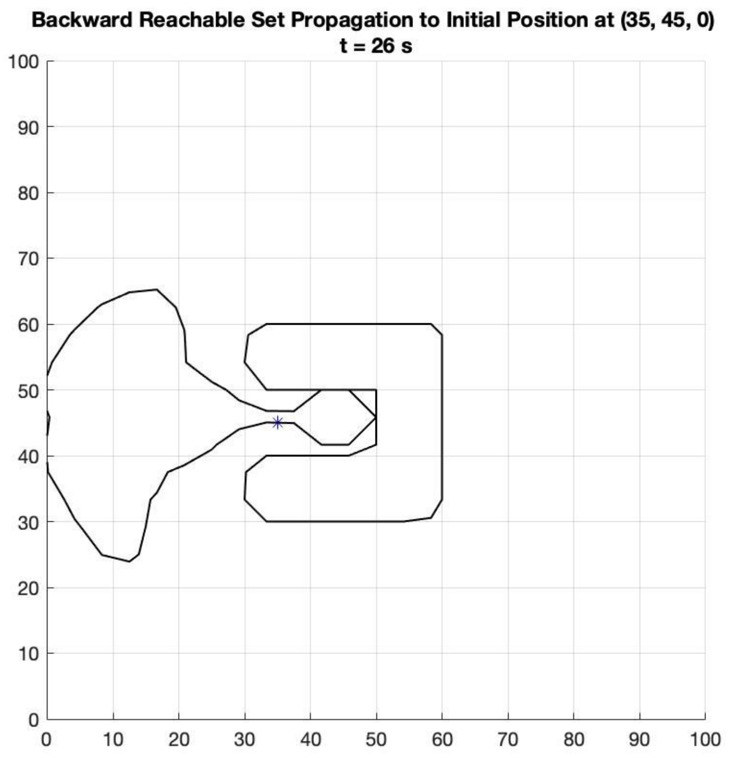


Figure 4.17: Backward reachable set propagation from target set to initial pose of UAV at $(35, 45, 0)$.

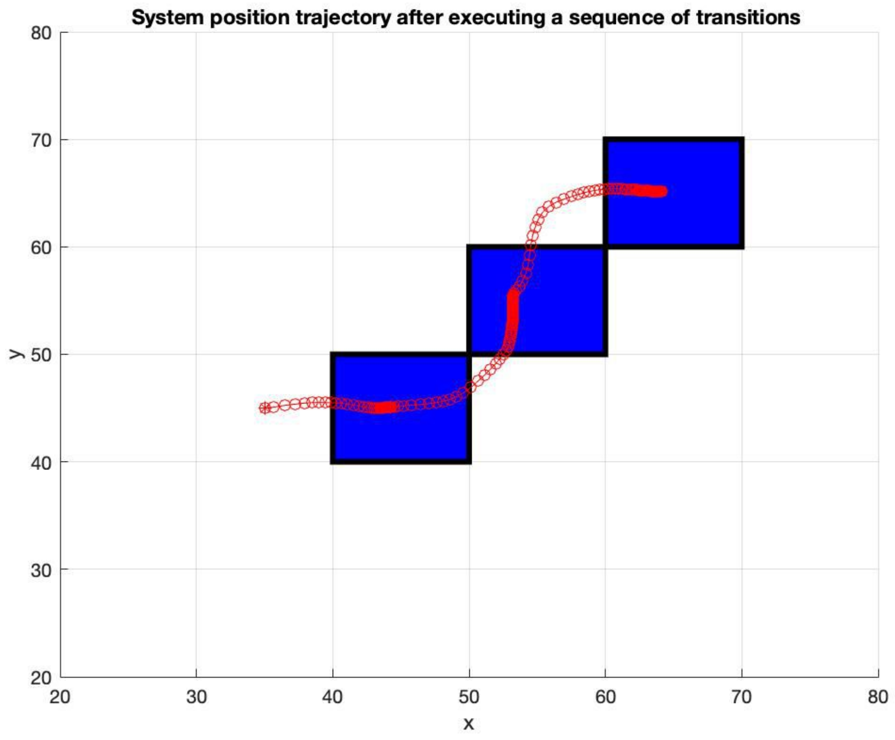


Figure 4.18: UAV trajectory obtained by the reachability analysis method after executing a sequence of transitions: forward, forward counter-clokwise turn, and forward clockwise turn.

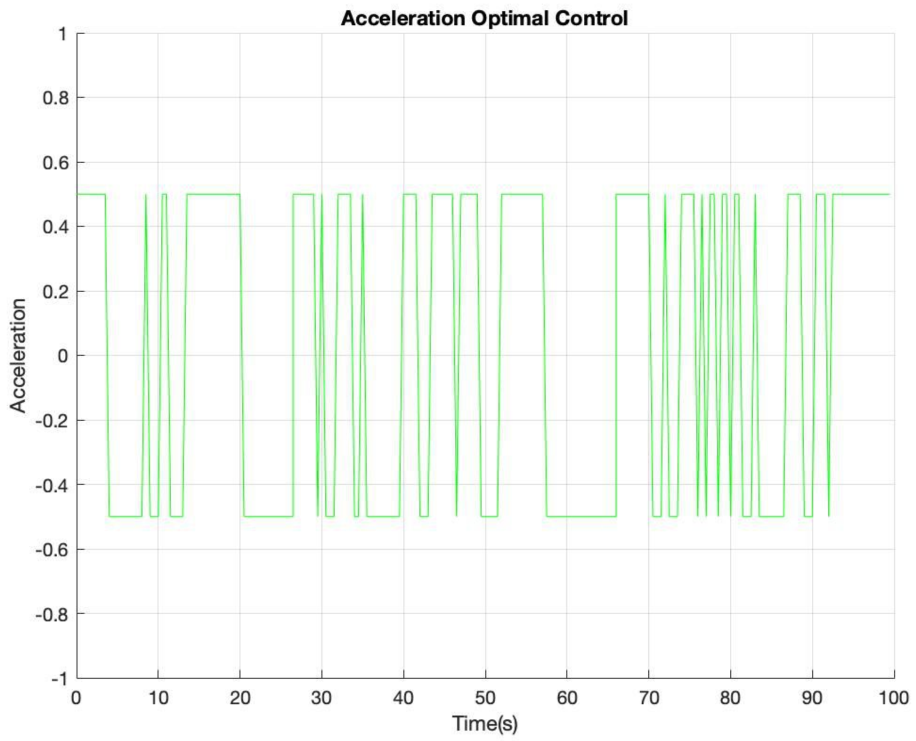


Figure 4.19: Optimal control for acceleration.

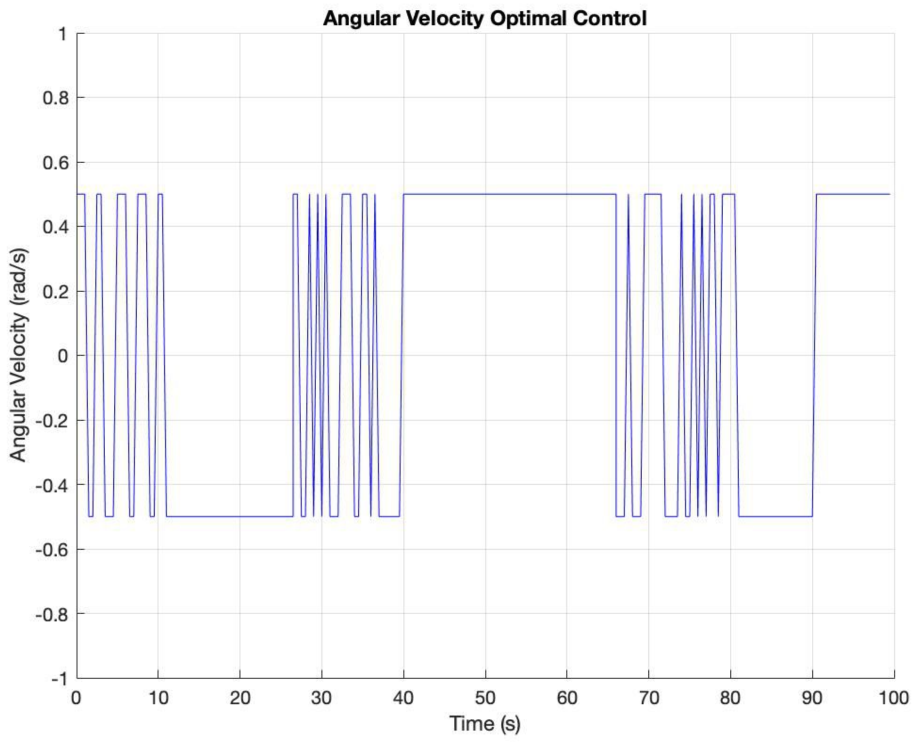


Figure 4.20: Optimal control for angular velocity.

4.5 Conclusions and Future Work

In this chapter, we constructed a high-level planner and controller for implementation in a fire fighting scenario. Our contributions include the RH framework modification combined with dynamic allocation, the algorithm for modifying the horizon definitions during synthesis, and the implemented simulation and results. The simulation demonstrated the method's ability to slow down the advancement of fire fronts towards the domain edge, providing a starting point of guaging the usefulness of automated UAVs in tackling fires before crews can arrive. The ability to slow down a starting fire by even half an hour to an hour (comparable to the maximum slowdown amounts in Scenarios 1, 2, 4, and 5) is a significant amount of time for ground crews to reach a location fast enough to effectively stop such a fire in its early phases. Expanding upon the receding horizon framework for reactive synthesis allowed us to expand the scope of this problem while integrating the method with dynamic allocation for assigning UAVs. Even with such an approach, numerous issues arose throughout the process that help highlight key difficulties moving forward when using reactive synthesis in the control of UAVs. First, the RH framework, when considering all initial conditions, still yields an excessively large controller (about 2 GB) after 8 hours of runtime, a significant hurdle for applying such a design when considering arbitrary static obstacle environments. Next, a simplified transition system was utilized which limited the total orientation space and interpreted UAV movement in only 2 dimensions, still far more restrictive than UAV movement in reality. Lastly, no constraints were used in considering the orientation of UAVs when dropping suppressant, which can significantly factor into how well the suppressant slows down an advancing fire front. Each of these points combine to exemplify the need for finer partitioning of possible transitions a UAV can take in 3D space (easily dependent on at least 3 full degrees of freedom), should reactive synthesize be used for UAV control. So while the reactive synthesis design is strong in enforcing the design constraints formally, the scope of its application is still limited per goal. For improvements on this problem as it was explored, multiple changes can be assessed. First, the size of the synthesized controller could be addressed through reformatting the outputted synthesized controller in each horizon. Currently, controllers are synthesized per initial condition in a horizon, but synthesizing a single controller for each horizon that includes all initial conditions can cut down on the total size of the generated finite state machine and possibly the synthesis time. Second, modifications to the synthesized controllers can be made to enforce desired orientations during suppressant drops. This will intuitively add to the size of the controllers but enable more accurate control of the UAVs for dropping suppressant in the correct direction. Lastly, direct coordination between synthesized controllers should be explored to control the frequency of suppressant drops on critical fires. The need for such is apparent by the "escaping" streak of fire present for all cases in Fig. 4.13. If the UAVs had spread out the times they dropped suppressant on the fire wrapping around the obstacle edge, the advancement of such would be hindered further since the UAVs would avoid refilling at the same time and better stretch out their resources. This effect could be achieved through modifying the allocation algorithm to optimize the total assignment of UAVs through a finite time horizon, perhaps achieved by expanding on the method presented in [18] and performing a type of tree search across sequential UAV assignment options and their associated costs.

Part II

Specification Inference from Formal Behavior

Chapter 5: Specification Inference via IRL

Specification inference serves a key role in robotics, enabling experts to demonstrate behavior as a natural way to convey task specifications, rules and norms. In many situations of practical interest, such as human motion in pedestrian settings and robot navigation, it is important to become aware of implicit norms that guide commonly seen motion patterns without the help of an expert providing these norms. The gained information, for instance, can be leveraged to better understand and predict motion patterns in pedestrian settings governed by known social norms.

One popular approach to addressing the challenge of specification inference is inverse reinforcement learning (IRL). In this approach, the demonstrator, operating in a stochastic environment, is assumed to attempt to optimize some unknown reward function over its trajectories [93]. The goal of IRL is to infer this reward function as it would serve to encode and generalize the goals of the demonstrator to new and unseen environments. However, the inference process can be challenging due to the properties of demonstrations. Demonstrations can be noisy and prone to error, unlabeled, context-dependent and ambiguous (i.e. partially ordered). Thus, the inference methods we are interested in need to be i) noise resistant, ii) unsupervised, and iii) computationally efficient.

In this work, our goal is to apply IRL to not only infer the unknown reward function but also the specification that best explains the agent's behavior given a sequence of demonstrations and a collection of specifications. We leverage LTL to write these specifications for greater interpretability and explanatory power in describing temporally extended (i.e. non-Markovian) patterns like safety and reachability rules with temporal dependencies among different goals.

In this chapter, we first describe the gaps and limitations of three state-of-the-art IRL methods tested to infer a social norm commonly practiced in pedestrian settings. We state this norm as "wait in line to get service". We present results from testing each of the chosen methods on demonstrations of this norm to draw performance comparisons. We then propose extensions to address the gaps of the inference method that showed the best performance and present results from our inference framework on different types of behavior demonstrations. To leverage the inference results to improve human motion prediction, we present a motion prediction framework that computes a posterior belief over the human's goal sequence and generates an occupancy grid for the human's future states using prior probabilities set according to the inference results. Finally, we conclude with a discussion of some of the limitations of our work and how they might be addressed in pedestrian applications, as well as suggestions for future research.

5.1 Prior Work

With the goal of learning specifications from expert demonstrations in stochastic settings, the underlying dynamics of the expert are modeled as a probabilistic automaton, denoted as PA. This characterization allows us to model the agent as choosing an action and the environment as sampling a state transition outcome. A probabilistic automaton is defined as follows.

Definition 5.1 *Let a probabilistic automaton (PA) be a tuple (S, s_0, A, δ), where S is the finite set of states, $s_0 \in S$ is the initial state, A is a finite set of actions, and δ specifies the transition probability of going from state s to state s' given action a, i.e. $\delta(s, a, s') = Pr(s'|s, a)$. A* ***demonstration****, ξ, is a sequence of (action, state) pairs starting from s_0. Thus, a demonstration of length $\tau \in \mathbb{N}$ is an element of $(A \times S)^\tau$.*

This system definition is adopted by each of the IRL methods described below.

5.1.1 Logic Based IRL

Logic Based IRL is an approach based on automata and logic based encodings of rewards [94][95] with the objective of translating them to LTL specifications. Given a set of specifications, the goal of this method is to infer an LTL specification from demonstrated behavior trajectories in the PA. To do so, this method finds the specification that minimizes the expected number of violations by an optimal agent compared to the expected number of violations by an agent applying actions uniformly at random [96]. An agent accrues violations every time its behavior deviates from the specification. The computation of the optimal agent's expected violations is done by applying dynamic programming on the product of the deterministic Rabin automaton [97], representing the LTL specification, and the state dynamics, modeled as a PA. This method has two main drawbacks, i) heavy run-time cost due to the curse of history and ii) lack of likelihoods over candidate specifications.

5.1.2 Traditional IRL

The goal of traditional IRL is to infer the reward function that best explains the actions of an agent operating in a MDP seeking to maximize its reward. The total reward accrued by the agent can be computed for each trajectory ξ over the PA as,

$$R(\xi) := \sum_{s \in \xi} r(s) = \theta.\mathbf{f}(s), \tag{5.1}$$

where s represents a state in the trajectory ξ, r is a reward map from states to reals, $r : S \to \mathbb{R}$, $\theta \in \mathbb{R}^n$ represents the preference in reaching or avoiding certain states, and $\mathbf{f}$ denotes the state features, $\mathbf{f}$: $S \to \mathbb{R}^n_{\geq 0}$.

Since there can be many reward functions that could explain the agent's behavior for a given set of demonstrations, one can apply the principle of maximum causal entropy [99] to find a policy consistent with the observed features from the expert demonstrations. Below are some definitions needed to state this principle.

Definition 5.1 *Let $X_{1:\tau} := X_1, ..., X_\tau$ denote a temporal sequence of $\tau \in \mathbb{N}$ random variables. The probability of a sequence $Y_{1:\tau}$ causally conditioned on sequence*

$X_{1:\tau}$ *is:*

$$Pr(Y_{1:\tau}||X_{1:\tau}) := \prod_{t=1}^{\tau} Pr(Y_t|X_{1:t}, Y_{1:t-1}) \tag{5.2}$$

The ***causal entropy*** *of* $Y_{1:\tau}$ *given* $X_{1:\tau}$ *is defined as,*

$$H(Y_{1:\tau}||X_{1:\tau}) := \mathbb{E}_{Y_{1:\tau},X_{1:\tau}}[-log(Pr(Y_{1:\tau}||X_{1:\tau}))] \tag{5.3}$$

The principle of maximum causal entropy seeks to find the policy whose action sequence $A_{1:\tau}$, maximizes the causal entropy $H(A_{1:\tau}||S_{1:\tau})$, conditioned on the state sequence $S_{1:\tau}$ and subject to feature matching constraints.

The learner learns form a dataset of N trajectories, $\mathcal{D} = \{\xi_1, \xi_2, ..., \xi_N\}$. Each trajectory $\xi_i = \{(s_1^{\xi_i}, a_1^{\xi_i}), (s_2^{\xi_i}, a_2^{\xi_i}), ..., (s_\tau^{\xi_i}, a_\tau^{\xi_i})\}$ is a state-action sequence of length τ. Given $\mathcal{D}$, the learner can compute the empirical feature expectation as,

$$\tilde{\mu}_k^{\mathcal{D}} := \frac{1}{N} \sum_{\xi \in \mathcal{D}} \sum_{t=1}^{h} \mathbf{f}_k(s_t^{\xi}). \tag{5.4}$$

The feature expectation given a policy π can be computed as,

$$\mu_k^{\pi}|_{\mathcal{D}} := \sum_{s \in S} P_{\mathcal{D}}(s_1 = s) \mu_k^{\pi}|_{s_1 = s}, \tag{5.5}$$

where $P_{\mathcal{D}}(s_1 = s) = N_1(s)/N$ is the maximum likelihood of the expert's initial state distribution, given that $N_1(s)$ is the number of trajectories with $s_1 = s$.

In addition to learning from a set of successful demonstrations $\mathcal{D}$, [98] also accounts for a set of failed demonstrations $\mathcal{F}$ and formulates the IRL problem as the following constrained optimization problem:

$$
\begin{aligned}
&\max_{\pi,w,z} H(A_{1:\tau}||S_{1:\tau}) + \sum_{k=1}^{K} w_k z_k - \frac{\lambda}{2}\|w\|^2 \\
&s.t.\ \ \mu_k^{\pi}|_{\mathcal{D}} = \tilde{\mu}_k^{\mathcal{D}}\ \ \forall k \\
&and\ \ \mu_k^{\pi}|_{\mathcal{F}} - \tilde{\mu}_k^{\mathcal{F}} = z_k\ \ \forall k \\
&and\ \ \sum_{a\in A} \pi(s,a) = 1\ \ \forall s \in S \\
&and\ \ \pi(s,a) \geq 0\ \ \forall s \in S, \forall a \in A
\end{aligned}
\tag{5.6}
$$

where λ is a constant. The objective in Eq. 5.6 seeks to maximize the causal entropy of π as well as the dissimilarity between π's feature expectations and the empirical expectations in $\mathcal{F}$. The third term serves to discourage large values of w.

Ziebart et. al [99] showed that solving Eq. 5.6 amounts to solving a soft Bellman equation:

$$log(\pi_\theta(a_t|s_t)) := Q_\theta(a_t, s_t) - V_\theta(s_t) \tag{5.7}$$

where

$$Q_\theta(a_t, s_t) := \mathbb{E}_{s_{t+1}}[V_\theta(s_{t+1})|s_t, a_t] + \sum_{k+1}^{K} \theta_k \mathbf{f}(s_t) \tag{5.8}$$

$$V_\theta(s_t) := ln \sum_{a_t} e^{Q_\theta(a_t,s_t)} = softmax_{a_t} Q_\theta(a_t, s_t), \tag{5.9}$$

where $\theta_k = w_k^{\mathcal{D}} + w_k^{\mathcal{F}}$ represents the sum of the weight vectors corresponding to the successful demonstrations and the failed demonstrations, respectively.

The major drawback of this approach is that despite successfully inferring the most probable reward function, this function is not able to capture temporal dependencies among different goals and cannot be easily interpreted. The IRL method described next addresses these drawbacks.

5.1.3 IRL from Non-Markovian Boolean Rewards

To explicitly capture temporal properties from demonstrations, the IRL approach proposed in [100] frames specification inference as the problem of learning Boolean specifications, which are able to express temporal dependencies among different goals. Thus, the rewards accrued by the demonstrator are captured as Boolean non-Markovian rewards to enable the handling of historical dependencies. This requires encoding the original MDP into a time unrolled MDP, which results in an algorithm with run-time exponential in the trace length. To address this computational complexity, the exponential time algorithm is translated into a polynomial time algorithm through the use of Reduced Ordered Binary Decision Diagrams (BDDs). Formally, the Boolean specifications and specification inference problem are stated as follows.

Definition 5.2 *A LTL specification φ is a subset of demonstrations,*

$$\varphi \subseteq (A \times S)^{\tau}. \tag{5.10}$$

Let a collection of specifications Φ define a concept class. Thus, $true := (A \times S)^{\tau}$, $\neg\varphi := true \setminus \varphi$, and $false := \neg true$.

Definition 5.3 *The specification inference problem is defined as a tuple (M, X, Φ, D) where $M = (S, s_0, A, \delta)$ is a probabilistic automaton, X is a (multi-) set of*

τ-length demonstrations drawn from an unknown distribution induced by a teacher attempting to satisfy some unknown specification within M, Φ is a concept class of specifications, and D is a prior distribution over Φ. A solution to (M, X, Φ, D) is:

$$\varphi^* \in \underset{\varphi \in \Phi}{\operatorname{argmin}} \, Pr(X \mid M, \varphi) \; . \; Pr_{\varphi \sim D}(\varphi), \tag{5.11}$$

where $Pr(X \mid M, \varphi)$ denotes the likelihood that the teacher would have demonstrated X give the task φ. Thus, the likelihood of multi-set i.i.d demonstrations, X, given a particular policy, $\pi(a|s) = Pr(a|s)$, is computed by:

$$Pr(X|M, \pi) = \prod_{\xi \in X} Pr(\xi|M, \pi). \tag{5.12}$$

One key aspect of the inference method is the machinery employed for embedding the full trace history into the state space, a process referred to as **unrolling**. This process is defined as follows,

Definition 5.4 *Let $M = (S, s_0, A, \delta)$ be a PA. The unrolling of M is a PA, $M' = (S', s_0, A, \delta')$, where*

$$S' = s_0 \times \bigcup_{i=0}^{\infty} (A \times S)^i \tag{5.13}$$

$$\xi_n = (s_0, ..., (a_{n-1}, s_n)) \tag{5.14}$$

$$\delta'(\xi_{n+1}, a, \xi_n) = \delta(s_{n+1}, a, s_n) \tag{5.15}$$

Given a non-Markovian reward R over τ-length traces, the unrolled PA can be endowed with the Markovian Reward in S',

$$r'(s_0, ..., (a_{n-1}, s_n)) := \begin{cases} R(s_0, ..., s_n) & \text{if } n = \tau \\ \\ 0 & \text{otherwise} \end{cases}, \tag{5.16}$$

$$\sum_{t=0}^{\infty} r'((s_0, a_0), ..., s_t) = R(s_0, ..., s_\tau). \tag{5.17}$$

Probabilistic automata can equivalently be characterized by $1\frac{1}{2}$ player games where each round has the agent choose and action and then the environment samples a state transition outcome. Let M be represented by the $1\frac{1}{2}$ player formulation and thus be encoded by a directed bipartite graph. For τ-length traces, this graph forms a decision tree $\mathbb{T}$ of depth τ. Hence, each τ-length trace over M associated with reward $R(\xi)$ corresponds to a leaf in $\mathbb{T}$.

For the purposes of specification inference, the non-Markovian reward corresponding to a specification φ is defined as,

$$R_\varphi(\xi) := \begin{cases} 1 & \text{if } \xi \in \varphi \\ \\ 0 & \text{otherwise} \end{cases}, \tag{5.18}$$

and the corresponding decision tree $\mathbb{T}$ becomes,

$$\mathbb{T}_\varphi : (A \times A_e)^\tau \to \{0, 1\}, \tag{5.19}$$

where A represents the actions of the system and A_e represents the actions of the environment. Using this reward definition in Eqs. (5.8) and (5.9), the policy that maximizes the causal entropy with respect to φ is obtained by computing the soft Bellman equation,

$$Q_\theta(a_t, \xi_t) = \mathbb{E}[V_\theta(\xi_{t+1})|\xi_t, a_t] \tag{5.20}$$

$$V_\theta(\xi_t) = \begin{cases} \theta.\varphi(\xi_t) & \text{if } t = \tau \\ softmax_{a_t} Q_\theta(a_t, \xi_t) & \text{otherwise} \end{cases} . \tag{5.21}$$

A downside of the dynamic programming scheme defined in Eqs. (5.20)-(5.21) over the unrolled tree $\mathbb{T}$ is its exponential blow-up. This issue is addressed by translating $\mathbb{T}$ to a reduced ordered probabilistic decision diagram, denoted as $\mathcal{T}$, via eliminating and combining isomorphic sub-graphs. To further reduce computational complexity, $\mathcal{T}$ is encoded as a Boolean predicate over an alternating sequence of action bit strings and coin flip outcomes (environment actions) determining whether the specification is satisfied or not,

$$\mathcal{T} : \{0, 1\}^n \to \{0, 1\}, \tag{5.22}$$

where $n := \tau log_2(|A \times A_e|)$. Thus, $\mathcal{T}$ is re-encoded as a reduced ordered BDD.

To summarize, the inference method consists in: i) selecting a specification from the concept class Φ, ii) constructing a BDD from the composition of the dynamical system with the specification, iii) computing the maximum causal entropy policy on the BDD, iv) encoding the demonstrations as bit-vectors, and v) computing the likelihood of the demonstrations using the encoded demonstrations and the policy.

5.2 Application to Learning Social Norms from Pedestrian Behavior

As previously mentioned, we are interested in applying specification inference to inferring social norms from demonstrations of pedestrian behavior. In particular, we are seeking to infer the commonly practiced social norm, "wait in line to get service" and compare the performance of the three IRL methods discussed with respect to learning this social norm.

To implement the three IRL methods discussed, we consider a simulated navigation domain in which the demonstrator navigates its environment to reach the *waiting area* while the *service area* is unavailable. Once the *service area* becomes available, the demonstrator proceeds to move to this area. The state space contains all possible (x, y) positions of the demonstrator. The action space is defined as $A = \{up, down, left, right, stay\}$. The workspace is set to be of dimension 20×20 with a region designated as the *service area* and another region designated as the *waiting area*.

For the logic based IRL method implementation, we defined the Boolean proposition, *serviceAvailable*, which is set to *true* when the *service area* is available, otherwise it is set to $false$. The action set is extended to $A = \{up, down, left, right, stay\} \times \{getToService, getInLine\}$. We also defined Boolean propositions $\{getToService$, $getInLine\}$ corresponding to each action (where, e.g., the proposition *getToService* is true whenever the agent's last action was to move towards the *service area*). The demonstrator is to satisfy the LTL specification,

$$\begin{aligned} \varphi = \Box((\bigcirc getInLine\, \mathcal{U}\; serviceAvailable) \\ \wedge\ (serviceAvailable \rightarrow \bigcirc getToService)). \end{aligned} \tag{5.23}$$

The "always" ($\Box$) temporal operator is included in φ (although it is not necessary to express the social norm in this form) since this method applies to specifications in persistent form only. A total of 20 demonstrator trajectories are generated to test this method. A sample demonstrator trajectory is obtained as follows,

$$\begin{aligned} \xi = (up, getInline), (right, getInLine), \\ (stay, getInLine), (up, getToService), \\ (up, getToService), (stay, getToService), \end{aligned} \tag{5.24}$$

and the corresponding trace of evaluations for the *serviceAvailable* proposition is $(false, false, false, true, true, true, true)$.

For the traditional IRL method implementation, we sample initial states from a uniform distribution over the state space for all experimental runs, including training and testing runs. Feature vectors are computed as discretized Euclidean distances between the demonstrator and the target into 5 possible values for the x and y directions, resulting in a feature vector $\mathbf{f}(s) \in \{0,1\}^{10}$ for any $s \in S$. In the case that the *service area* is unavailable, the weighting vectors for the successful demonstrations $w^{\mathcal{D}}$ are set to provide larger rewards to closer distances between the demonstrator and the *waiting area*, with the demonstrator receiving the largest reward when it arrives at the *waiting area*. For failed demonstrations, $w^{\mathcal{F}}$ is set to provide large reward values to large distances between the demonstrator and the *waiting area*. This choice of weighting vectors assumes that we have access to completely failed demonstrations and successful demonstrations that show the complete desired behavior. As soon as the *service area* becomes available, the reward function for successful demonstrations is redefined by providing the largest reward to the closest distance between the demonstrator and the *service area*. Similarly, the reward function for failed demonstrations is redefined by providing large rewards to larger distances between the demonstrator and the *service area*.

For the implementation of the last IRL method discussed, we also defined Boolean propositions to denote the *waiting area* and the *service area*. We generated 20 expert demonstrations and tested this approach on a set of candidate specifications involving the defined Boolean propositions. For all the methods, we assume that the demonstrator is able to use sensors to evaluate the truth value of the defined propositions about the regions of interest. More details about the implementation of this method and the extensions proposed will be discussed in the next section.

Table 5.1 summarizes the output type, run-time and drawbacks of each IRL approach tested. Based on the obtained results, we can observe that the most effective method in terms of the type of output provided and the average run-time is the method that uses Non-Markovian Boolean rewards. This method not only obtains a specification but also computes likelihoods over the candidate specifications. However, there are questions not addressed by this work that we will answer in the next section. These questions include: i) how should the hypothesis space of candidate specifications be generated?, ii) can we systematically perturb the specification to make it more likely?, iii) how to set prior probabilities for the candidate specifications?, and iv) how should we leverage inference results to improve motion prediction?.

IRL Method	Output	Avg. Runtime	Main Drawback(s)
Logic Based IRL	Specification that minimizes violation cost by an optimal agent.	Average run-time of 120 minutes for demonstration set of size 20.	Heavy run-time cost and no specification likelihoods provided.
Traditional IRL	The most probable reward function.	Average run-time of 110 and 330 minutes for demonstration sets of sizes 5 and 50, respectively.	Cannot represent temporally complex norms.
IRL from Non-Markovian Boolean Rewards	Posterior distribution over candidate specifications.	Average run-time of 7 minutes for demonstration set of size 20.	Does not address questions about selection of specifications and prior probabilities.

Table 5.1: Performance comparison of three types of IRL methods on inferring the social norm "wait in line to get service" with respect to output type, average run-time and main drawbacks.

5.3 Inference Framework

Building on the IRL method presented in [100], we propose the following extensions: i) generating training data from a selected specification in LTL and a dynamical system modeled as an MDP, ii) constructing the initial hypothesis space of specifications and systematically perturbing the specifications according to their likelihood estimates (we will refer to this process as specification generation and refinement), iii) computing the most likely specification using the likelihood estimates and the constructed prior probabilities. From the most likely specification, we derive the most likely goal sequence the demonstrator exhibits and set goal prior probabilities according to these sequences to be used for human motion prediction. Fig. 5.1 shows a diagram of the overall inference process presented in [100] and the extensions proposed in this work (highlighted in blue).

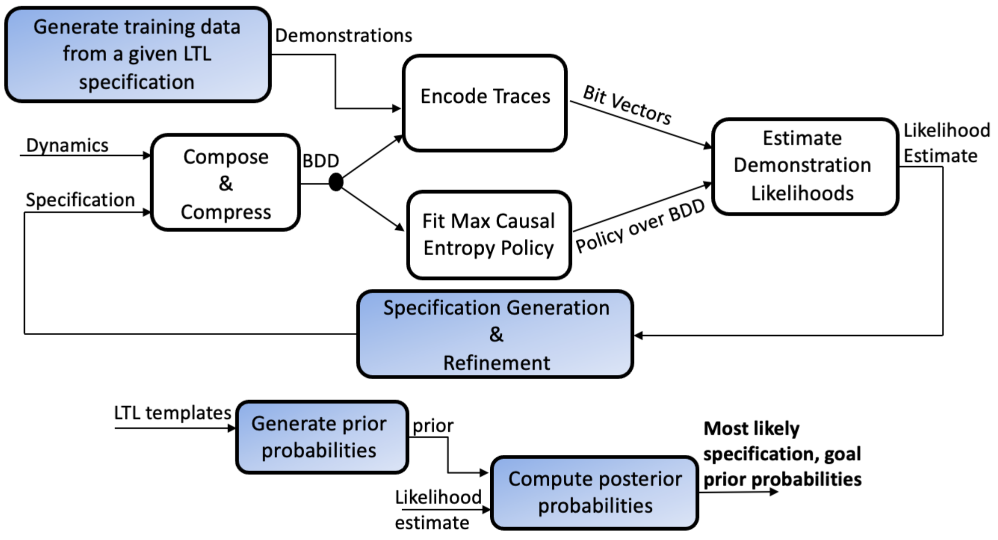

Generate training data from a given LTL specification
Demonstrations
Encode Traces
Bit Vectors
Estimate Demonstration Likelihoods
Likelihood Estimate
Dynamics
Specification
Compose & Compress
BDD
Fit Max Causal Entropy Policy
Policy over BDD
Specification Generation & Refinement
LTL templates
Generate prior probabilities
prior
Compute posterior probabilities
Likelihood estimate
Most likely specification, goal prior probabilities

Figure 5.1: Diagram of the IRL method presented in [100] and the extensions proposed highlighted in blue. The specification inference process begins with a set of candidate specifications that go through a refinement process. The desired output is the most likely specification computed by combining the prior probabilities and the likelihood estimates.

The first extension proposed facilitates the automatic generation of trajectories satisfying the specification we seek to learn. This process is done following a common framework within behavior synthesis from LTL specifications and a system model. In the following section, we describe the system model used to represent human motion behavior throughout this chapter.

5.3.1 Human Modeling

We model human walking behavior as an MDP with unknown transition probabilities in order to capture the unknown stochastic nature of human motion (e.g., speed, smoothness), which can vary significantly among different individuals [102]. More precisely, the human is modeled as a discrete-time stochastic control system defined as,

$$\Sigma = (X, U, \varsigma, f), \tag{5.25}$$

where $X \subseteq \mathbb{R}^n$ is the state space of the system, $U \subseteq \mathbb{R}^m$ is the input space of the system, ς is a sequence of independent and identically distributed (i.i.d.) random variables from a sample space Ω to the set V_ς and $f : X \times U \times V_\varsigma \to X$ is a measurable function characterizing the state evolution of Σ. The evolution of the state of Σ for an initial state $x(0) \in X$ and an input sequence $u(k) : \Omega \to U, k \in \mathbb{N}$ is described as:

$$x(k+1) = f_H(x(k), u(k), \varsigma(k)). \tag{5.26}$$

Let x represent the state of the human and n be the dimension of the human state space. This state could correspond to positions and velocities of the human. In this work, we define the evolution of the human state as,

$$\begin{aligned} x(k+1) &= \begin{bmatrix} h_x(k+1) \\ h_y(k+1) \end{bmatrix} \\ &= \begin{bmatrix} v_H(k)\cos\phi_H(k) + \varsigma_x(k) \\ v_H(k)\sin\phi_H(k) + \varsigma_y(k) \end{bmatrix}, \end{aligned} \tag{5.27}$$

where h_x and h_y are the planar coordinates of the human, v_H is the velocity, ϕ_H is the heading and ς is the additive noise, drawn according to a uniform distribution and used to model the stochasticity of the human. In this work, we assume that the human's intents and policies (i.e., goals and sequences of goals) are guided by social norms known to the human. We can formally express such normative rules as logic-based specifications using formulae in LTL.

5.3.2 Generation of Demonstrations

In general, for verification and synthesis purposes, the required LTL properties are translated into a deterministic finite automaton (DFA) $\mathcal{A}_\varphi$ over the alphabet $\mathbf{\Sigma}_a$ (formed by a set of atomic propositions AP), such that the set of infinite words satisfying the formula φ is equal to the set of infinite words accepted by the DFA $\mathcal{A}_\varphi$, denoted as $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$. Given an LTL formula, we can synthesize policies for the human over an abstract representation of the MDP that models the human's motion (as described in Eqs. 5.25-5.27). The goal of synthesis is to compute a policy ρ that maximizes the probability that a state trajectory of Σ satisfies an LTL formula over a finite time horizon T, denoted as $\mathbb{P}(w_f \in \mathcal{L}(\mathcal{A}_\varphi)\ s.t\ |w_f| \leq T+1)$. We employ the abstraction-based synthesis method proposed in [103] to generate satisfying trajectories with respect to social norms expressed as LTL formulas. The synthesis method from [103] abstracts the continuous-space MDP into a finite MDP system and synthesizes a policy using a model-free reinforcement learning scheme.

The main benefit of this approach is that it provides probabilistic closeness guarantees between the resulting behavior of the finite MDP and its continuous-space counterpart. Thus, the Markov policy obtained enforces the given formula φ over Σ with the probability of satisfaction within a guaranteed threshold from the unknown optimal probability. A limitation of this approach is that it only works with a fragment of LTL properties known as syntactically co-safe linear temporal logic (scLTL). This fragment of LTL requires the negation operator ($\neg$) to only occur before atomic propositions. The reason for limiting the space of LTL properties to scLTL formulas is that even though scLTL are defined over infinite words, their satisfaction can be guaranteed in finite time [104]. For our purposes, we consider social norms that can adopt the form of scLTL properties and assume this representation to evaluate finite horizon satisfaction. Fig. 5.2 shows a diagram summarizing the policy synthesis process from a given LTL specification.

Once satisfying trajectories are obtained for a given LTL specification and dynamical system, we proceed to discretize the trajectories and use them as expert demonstrations in the specification inference scheme. The next extension to be discussed addresses the questions previously posed about the construction of the hypothesis space of candidate specifications.

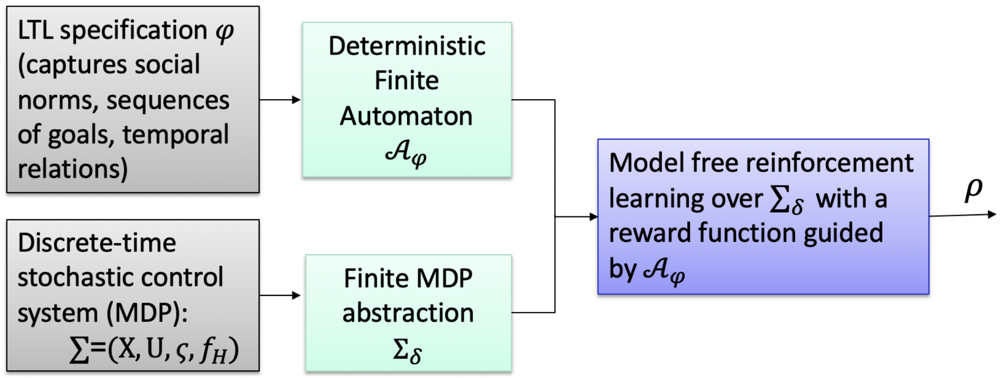

LTL specification φ (captures social norms, sequences of goals, temporal relations)
Discrete-time stochastic control system (MDP): $\Sigma=(X, U, \varsigma, f_H)$
Deterministic Finite Automaton $\mathcal{A}_\varphi$
Finite MDP abstraction Σ_δ
Model free reinforcement learning over Σ_δ with a reward function guided by $\mathcal{A}_\varphi$
ρ

Figure 5.2: Diagram of the synthesis process [103] resulting in a policy ρ that maximizes the probability that a system trajectory satisfies φ over a finite time horizon. This policy is used to generate demonstration trajectories of a human performing a social norm encoded by φ.

5.3.3 Hypothesis space construction

To apply the inference method described in [100], we restrict our hypothesis space to a finite set of Past Linear Temporal Logic (PLTL) templates. PLTL is a formal language analogous to LTL with the difference that it serves to express properties over finite traces in past time. The temporal operators of PLTL include: *yesterday* ($\bigcirc^{-1}$), analogous to *next*, *historically* ($\Box^{-1}$), analogous to *always*, *once* ($\Diamond^{-1}$), analogous to *eventually*, and *since* (S), analogous to *until*. We refer the reader to [101] for more details on the formalism of PLTL.

We propose that any specification can be composed by the following elements: i) a selection of a number of free propositions n_P, denoted as $\mathbf{p} \in AP$, where AP is a set of atomic propositions, ii) a selection of a PLTL template involving a combination of temporal and Boolean operators, and iii) a selection of a number of conjunctions N, initially set to 1. Thus, the first step in the hypothesis construction is to select a PLTL template P. Once P is selected, it is instantiated with a selection of n_P propositions $\mathbf{p}$. For example, consider a set of propositions $AP = \{\mathit{red}, \mathit{yellow}, \mathit{brown}, \mathit{blue}\}$. Consider a formula consisting of a single conjunct with template P. We can instantiate P to "historically" with $\mathbf{p} = [\mathit{red}]$ to obtain the formula $\varphi = \Box^{-1}(\mathit{red})$. If the template selected is S and $\mathbf{p} = [\mathit{red},\ \mathit{yellow}]$, then the formulas obtained are $\varphi_1 = (\mathit{red}\ S\ \mathit{yellow})$ and $\varphi_2 = (\mathit{yellow}\ S\ \mathit{red})$. If $N = 2$, $P = \mathit{historically}$ and $\{\mathbf{p}\} = \{[\mathit{red}], [\mathit{blue}]\}$, we obtain the formula $\varphi = \Box^{-1}(\mathit{red}) \ \wedge \Box^{-1}(\mathit{blue})$. Thus, each φ is generated by choosing a PLTL template P, the number of conjunctions N, and the proposition instantiations, $\mathbf{p}$. For $N = 1$, the number of possible instantiations that can be generated from a set AP with n propositions and number of free propositions n_P, is computed as $\frac{n!}{n_P!(n-n_P)!}$. The generated formulas form a hypothesis space for the concept class of specifications Φ. In this way, Φ is systematically constructed from a given set of predefined templates and propositions. See Table 5.2 for a summary of the PLTL templates used in this work.

To refine the initial hypothesis space, we apply a modification algorithm that perturbs a subset of specifications in Φ. This subset Φ' contains all the specifications for which the inference process obtains higher likelihoods compared to the *True* specification. This ensures that for each specification in Φ', it is more likely that the teacher would have demonstrated behavior given the specification than given that the agent applies actions uniformly at random. The refinement process combines all the specifications in Φ' with the same template into a new specification via conjunctions. If the refined specification does not obtain a higher likelihood compared to *True*, we can modify it by removing the conjunct with the lowest like-

Template	n_P	Formula	Meaning
φ_{avoid}	1	$\Box^{-1}(\neg p_i)$	p_i was not true throughout the entire trace.
$\varphi_{eventually}$	1	$\Diamond^{-1} p_i$	p_i occurred at some point in the trace.
φ_{since}	2	$p_i S p_j$	p_i has been true since a time when p_j was true.
$\varphi_{response}$	2	$\Box^{-1}(p_i \rightarrow \Diamond^{-1} p_j)$	At all times that p_i occurred, p_j was true at some point after.
$\varphi_{eventuallysince}$	2	$(\Diamond^{-1} p_i) S p_j$	p_i occurred at some point in the trace since a time when p_j was true.
$\varphi_{implication}$	2	$\Diamond^{-1} p_i \rightarrow \bigcirc^{-1} p_j$	If p_i was true at some point in the trace, then p_j was true at some previous time.

Table 5.2: PLTL templates considered in this work. n_P represents the number of free propositions for each template.

lihood estimate. We repeat this process of removing a conjunct from the formula at a time to ensure that the specification is systematically perturbed until it obtains a higher likelihood compared to *True* or there are no more conjuncts to remove. The refinement process is shown in Algorithm 3.

In the next section, we describe the prior function that serves to assign a prior to each specification.

Algorithm 3: Φ Modification During Inference

```
1: procedure REFINEMENT($\Phi'$, $\Phi_P$, $\mathcal{E}$, $P$, $min_P$)    ▷
   Compute a refined hypothesis space of candidate specifications $\Phi'$ based on $\Phi_P$
   and the corresponding likelihood estimates $\mathcal{E}$, template $P$ and lowest likelihood
   estimate $min_P$.
2:     for $\varphi_P \in \Phi_P$ do
3:         $\varepsilon = \mathcal{E}(\varphi_P)$
4:         Add conjunct
5:         if $\varepsilon > \mathcal{E}(True)$ and $N(\varphi_P) = 1$ then
6:             $\varphi_{new} = \varphi_p \wedge \varphi_{new}$
7:         end if
8:         Remove conjunct
9:         if $\varepsilon < \mathcal{E}(True)$ and $N(\varphi_P) > 1$ then
10:            for $conjunct \in \varphi_p$ do
11:                if $\mathcal{E}(conjunct) = min_P$ then
12:                    $\varphi_{new}$ = Remove $conjunct$ from $\varphi_P$ if $N(\varphi_P) > 1$
13:                    break
14:                end if
15:            end for
16:        end if
17:        Add $\varphi_{new}$ to $\Phi'$
18:    end for
19: end procedure
```

5.3.4 Prior Function

We now define a prior function over the hypothesis space Φ to serve as a preference module for the system designer. We define a categorical distribution with weights $w_P \in \mathbb{R}^k$ over the k possible PLTL templates and a categorical distribution with weights $w_p \in \mathbb{R}^{|AP|}$ for all propositions $p \in AP$. The system designer can use w_P to set preferences for inferring certain types of templates over others. Likewise, the designer can set w_p to give preference to certain types of propositions. For example, propositions that represent places of interest for planning problems may require the designer to favor these propositions. The number of conjunctions N can be assigned a probability according to a geometric distribution with a decay rate of λ. The reasoning behind this is that mining low-complexity specifications (those with fewer number of conjunctions) should be preferable over mining long and convoluted specifications.

Combining all the elements that completely specify a formula φ, we now state the full prior function for φ as follows:

$$Pr(\varphi) = Pr(P)Pr(N)Pr(\{\mathbf{p}\}), \tag{5.28}$$

where $Pr(P)$ and $Pr(N)$ are calculated using categorical and geometric distributions, respectively. $Pr(\{\mathbf{p}\})$ is calculated by the average categorical weight, w_P, over all propositions,

$$Pr(\{\mathbf{p}\}) = \frac{\sum_{\mathbf{p}\in\{\mathbf{p}\}} \sum_{p\in\mathbf{p}} w_p}{\sum_{\mathbf{p}\in\{\mathbf{p}\}} |\mathbf{p}|}. \tag{5.29}$$

For example, for $N = 2$ and $\{\mathbf{p}\} = \{[red, yellow], [yellow, blue]\}$, and $w_{red} = \frac{1}{4}$, $w_{yellow} = \frac{1}{2}$ and $w_{blue} = \frac{1}{4}$, $Pr(\{\mathbf{p}\}) = \frac{5/4}{4} = \frac{5}{16}$.

5.4 Experimental Setup

To evaluate the inference process, we define 4 different scenarios in which the demonstrator behavior can be expressed by an LTL specification. We set the bounds for the state space of the human model as $h_x \in [0.0, 20.0]$ and $h_y \in [0.0, 20.0]$ and we set the bounds for the input to the system, $\phi_H \in [-\pi, \pi]$. We set $v_H = 1.4m/s$. The co-variance matrix of the additive noise is set to

$$\Sigma = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}. \tag{5.30}$$

We set the discretization parameter for the system states to 0.5 and the discretization parameter for the system input to 0.05. Table 5.3 provides a summary of the scenario definitions and the corresponding specifications we seek to infer from each scenario.

The implementation of the policy synthesis from a given LTL specification is done via the open source tool AMYTISS [105]. The obtained policy is then used to generate trajectories satisfying the given specification within a maximum of 20 time-steps. The resulting continuous trajectories are discretized and provided as demonstrations to the inference process. Fig. 5.3 - 5.6 shows sample trajectories for each scenario implemented.

Scenario	PLTL specification
Wait in line if service area is unavailable.	$\Diamond^{-1}(\neg\ SA_{accessible}\ \wedge\ \mathrm{WA})$
Wait in line if service area is unavailable, then get service once service area becomes available, then proceed to exit.	$\Diamond^{-1}(\neg\ SA_{accessible}\ \wedge\ \mathrm{WA})$ $\wedge\ (\Diamond^{-1}(SA_{accessible}\ \wedge\ SA) \rightarrow \bigcirc^{-1}\mathrm{WA})$ $\wedge\ (\Diamond^{-1}\ \mathrm{Exit} \rightarrow \bigcirc^{-1}SA)$
Visit area 1, then area 2, then area 3	$(\Diamond^{-1}G3 \rightarrow \bigcirc^{-1}G2)$ $\wedge\ (\Diamond^{-1}G2 \rightarrow \bigcirc^{-1}G1)$
Avoid obstacle areas	$(\Box^{-1}\neg O)$

Table 5.3: Chosen PLTL specifications for each scenario. In the first two scenarios, we denote the service area as SA and the waiting area as WA. $SA_{accessible}$ is used to denote whether or not the service area is accessible. In the third scenario, $G1$, $G2$ and $G3$ denote area 1, area 2, and area 3, respectively. O denotes obstacle areas.

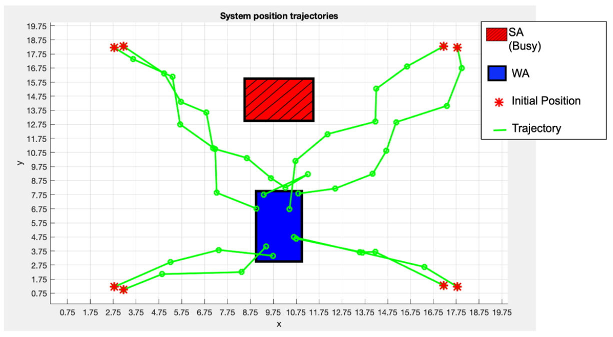


Figure 5.3: Sample trajectories for scenario with specification, “wait in line if service area is unavailable”.

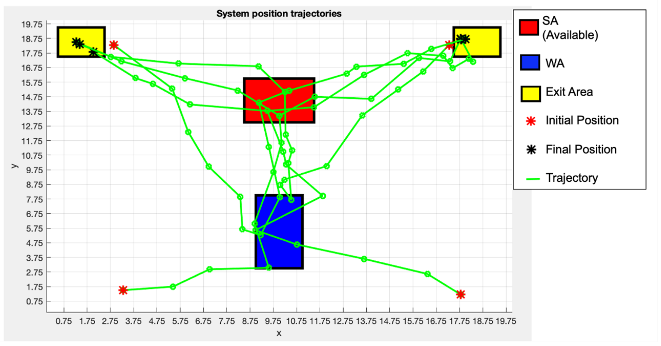


Figure 5.4: Sample trajectories for scenario with specification, “wait in line if service area is unavailable, then get service once service area becomes available, then proceed to exit.”.

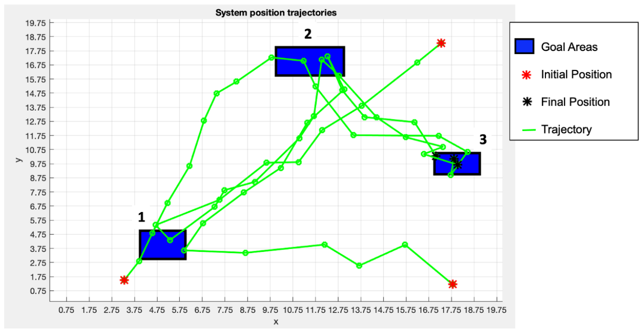


Figure 5.5: Sample trajectories for scenario with specification, “visit area 1, then area 2, then area 3”.

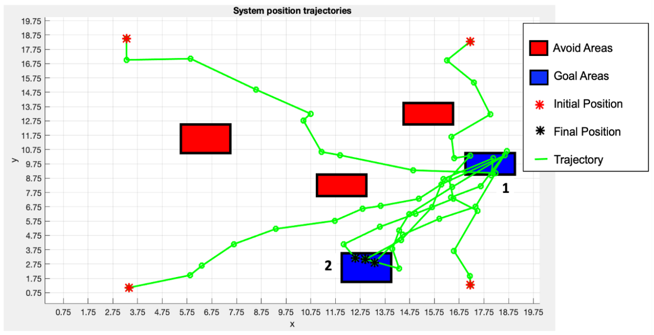


Figure 5.6: Sample trajectories for scenario with specification, “avoid obstacle areas”.

5.5 Inference Results

For all scenarios, we set $\lambda = 0.8$ and the template weights as $w_{avoid} = 0.17$, $w_{eventually} = 0.17$, $w_{since} = 0.21$, $w_{response} = 0.12$, $w_{eventuallysince} = 0.12$, $w_{implication} = 0.21$. Doing so favors *implication* and *since* templates over other templates to give preference to specifications that capture temporal dependencies among goals. All other templates are given similar weights with slightly larger values given to simpler templates.

Scenario 1

For the first scenario, we construct the hypothesis space Φ over $AP = \{SA, WA\}$ with proposition weights $w_{SA} = 1/2$ and $w_{WA} = 1/2$. In this scenario, we assume that the service area is unavailable by default, which implies $\neg\ SA_{accessible}$ evaluates to $True$, $(\neg\ SA_{accessible}\ \wedge\ WA)$ evaluates to the truth value of WA and $(\neg\ SA_{accessible}\ \wedge\ SA)$ evaluates to the truth value of SA at all times. Using the templates defined in Table 5.2 and AP, 14 specifications were initially generated. Fig. 5.7 shows the relative likelihoods for each of these specifications. From these results, we can observe that there are only two specifications for which we obtained positive values for the relative likelihoods, which implies that only these specifications are more likely than random behavior. Since the resulting specifications have different templates, no refinement is needed and the most likely specification is computed using the corresponding priors.

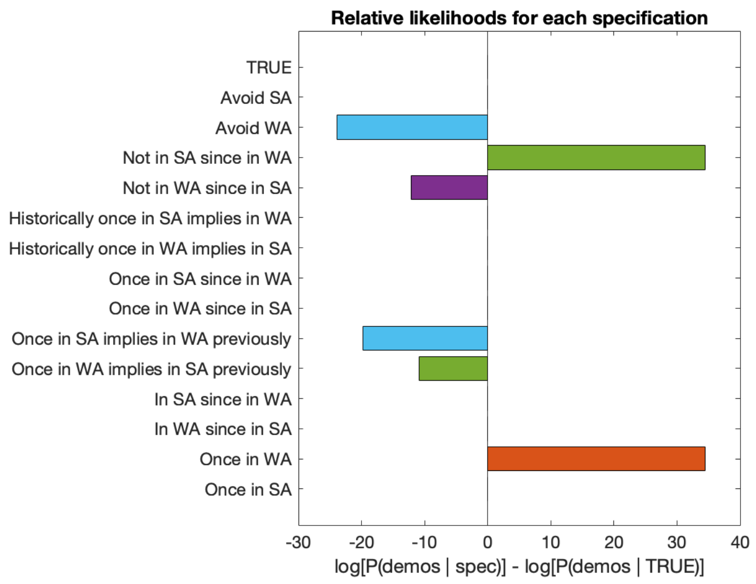


Figure 5.7: Inference results for scenario, "wait in line if service area is unavailable". Positive values correspond to specifications that are more likely than random behavior.

Scenario 2

For the second scenario, we construct the hypothesis space Φ over

$$AP = \{SA, WA, SA_{accessible}\}, \tag{5.31}$$

with proposition weights $w_{SA} = 1/3$, $w_{WA} = 1/3$, and $w_{SA_{accessible}} = 1/3$. Using the templates defined in Table 5.2 and AP, 36 specifications were initially generated. Fig. 5.8 shows the relative likelihoods for each of these specifications. We can observe that there are 14 specifications for which we obtained positive values for the relative likelihoods. Then, we proceed to combine the specifications with the same templates and we obtain the the relative likelihoods for the refined specifications shown in Fig. 5.9. From these results, we note that the specification with the *eventuallysince* template does not obtain a positive value for the relative likelihood; thus we proceed to remove the conjunct with the least likelihood and compute the relative likelihoods again as shown in Fig. 5.10. At this point, all specifications have positive relative likelihood values and no further refinement is needed.

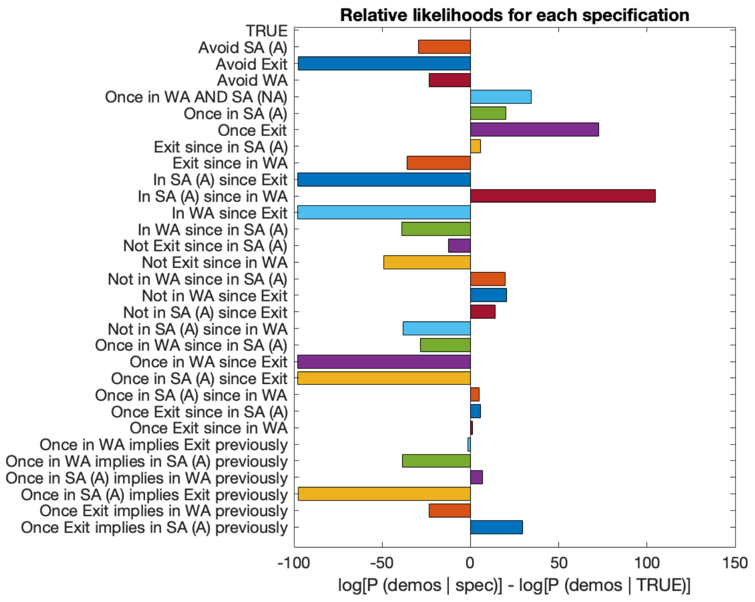


Figure 5.8: Inference results for scenario, "wait in line if service area is unavailable, then get service once service area becomes available, then proceed to exit." For conciseness, we do not include specifications with zero relative likelihood values. We use SA (A) to denote SA and $SA_{accessible}$ and SA (NA) to denote SA and $\neg SA_{accessible}$.

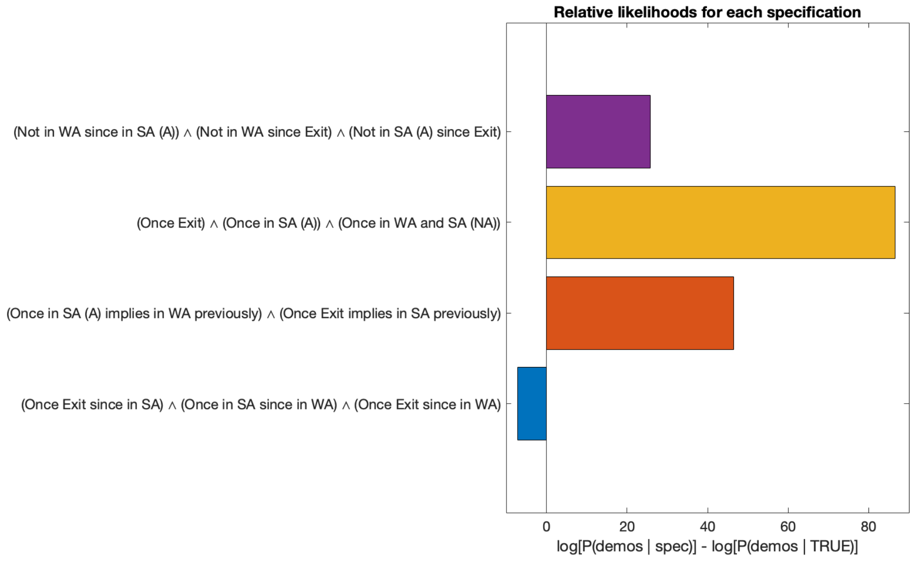


Figure 5.9: Inference results from refined specifications after 1 iteration of refinement for scenario, "wait in line if service area is unavailable, then get service once service area becomes available, then proceed to exit."

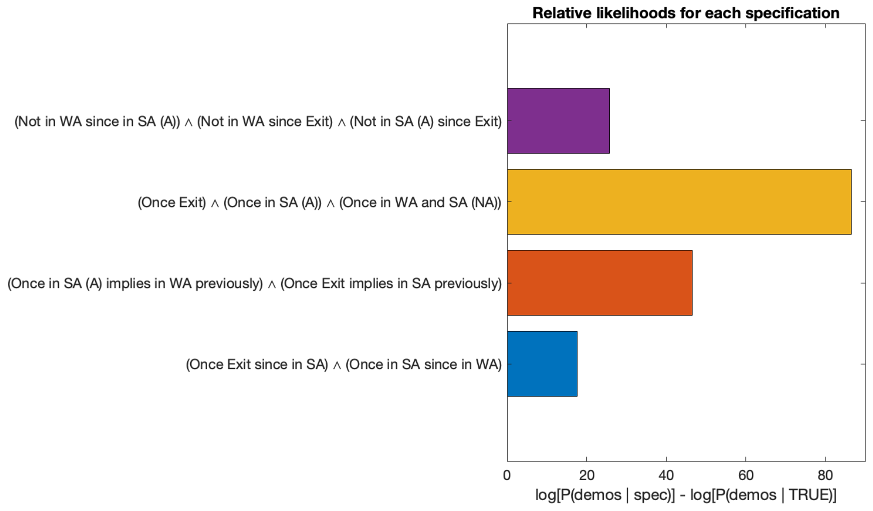


Figure 5.10: Inference results from refined specifications after 2 iterations of refinement for scenario, "wait in line if service area is unavailable, then get service once service area becomes available, then proceed to exit."

Scenario 3

For the third scenario, we construct the hypothesis space Φ over

$$AP = \{G1, G2, G3\}, \tag{5.32}$$

with proposition weights $w_{G1} = 1/3$, $w_{G2} = 1/3$, and $w_{G3} = 1/3$. Using the templates defined in Table 5.2 and AP, 36 specifications were initially generated. Fig. 5.11 shows the relative likelihoods for each of these specifications. We can observe that there are 8 specifications for which we obtained positive values for the relative likelihoods. Then, we proceed with the refinement process and we obtain the the relative likelihoods for the refined specifications shown in Fig. 5.12.

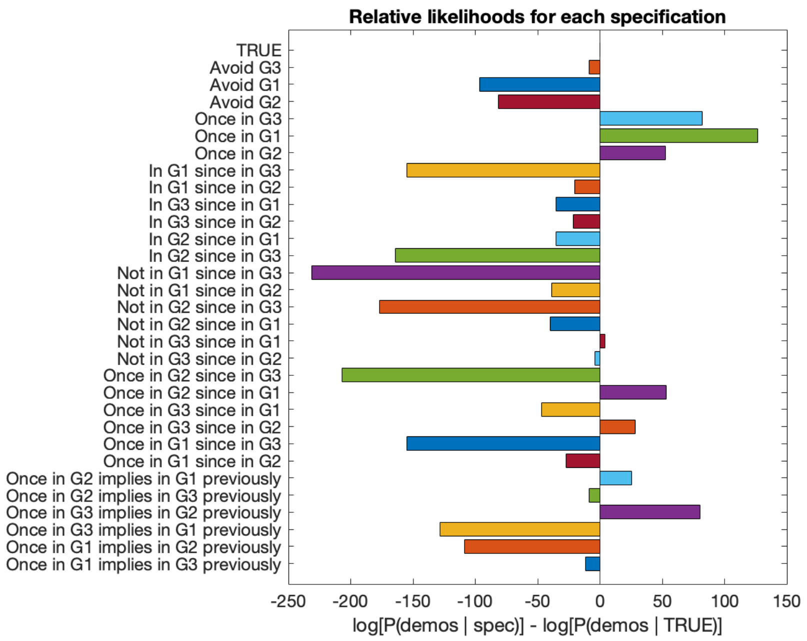


Figure 5.11: Inference results for scenario, "visit area 1, then area 2, then area 3." We denote area 1 as G1, area 2 as G2, and area 3 as G3.

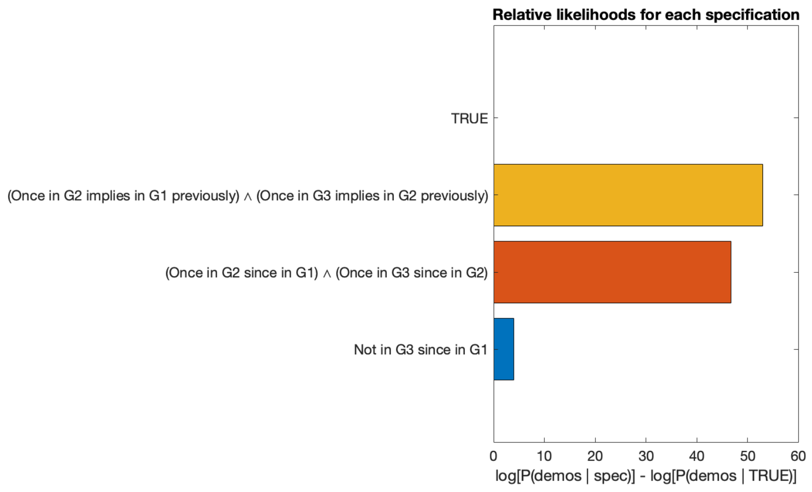


Figure 5.12: Inference results from refined specifications after 1 iteration of refinement for scenario, "visit area 1, then area 2, then area 3."

Scenario 4

For the last scenario, we construct the hypothesis space Φ over

$$AP = \{G1, G2, O\}, \tag{5.33}$$

with proposition weights $w_{G1} = 1/3$, $w_{G2} = 1/3$, and $w_O = 1/3$. We use O to denote to areas that should be avoided. Using the templates defined in Table 5.2 and AP, 36 specifications were initially generated. Fig. 5.13 shows the relative likelihoods for each of these specifications. We can observe that there are 9 specifications for which we obtained positive values for the relative likelihoods. Then, we proceed with the refinement process and we obtain the the relative likelihoods for the refined specifications shown in Fig. 5.14.

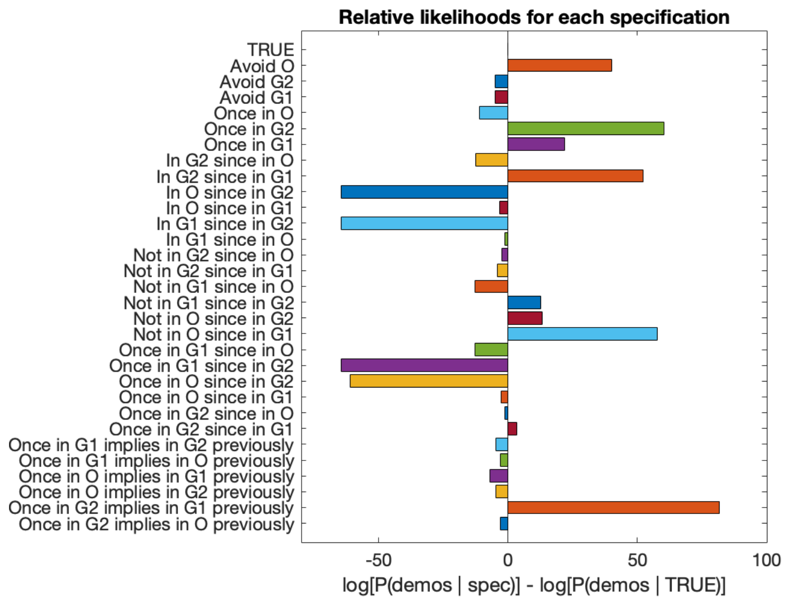


Figure 5.13: Inference results for scenario, "avoid obstacle areas." We denote area 1 as G1, area 2 as G2, and obstacle areas as O.

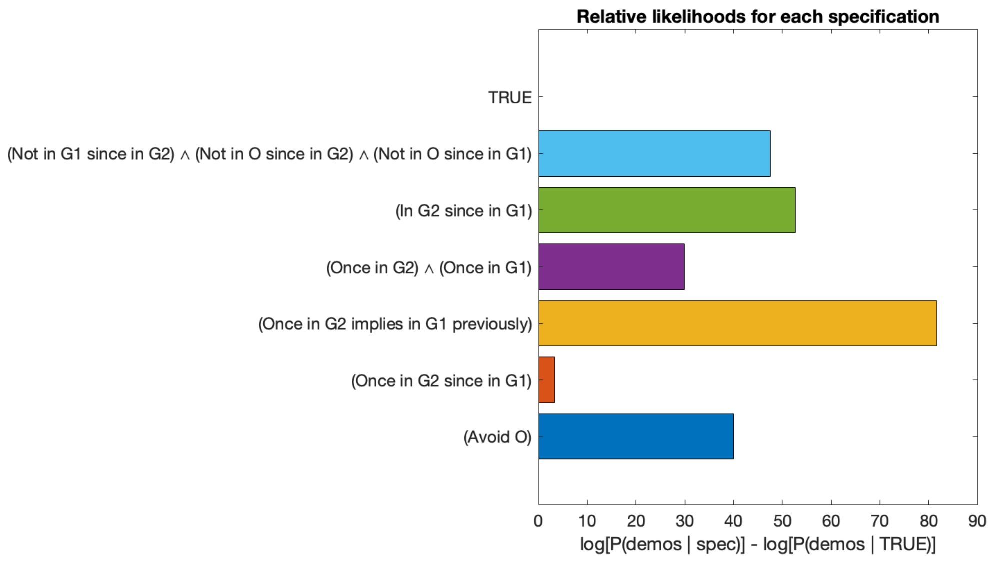


Figure 5.14: Inference results from refined specifications after 1 iteration of refinement for scenario, "avoid obstacle areas."

From the inference results, we can derive the goal sequence from the most likely specification in each scenario and summarize these results for all scenarios in Table 5.4. Note that for all scenarios, except for the last one, the most likely specification

Scenario	**Most Likely Specification**	**Total Time**	**Goal Sequence**
Wait in line if service area is unavailable.	$\neg SA_{accessible} \wedge (\neg(\text{SA})\ S\ (\text{WA}))$	15.5 s	WA
Wait in line if service area is unavailable, then get service once service area becomes available, then proceed to exit.	$(\Diamond^{-1}(SA_{accessible} \wedge SA) \rightarrow \bigcirc^{-1}(\text{WA} \wedge \neg SA_{accessible})) \wedge (\Diamond^{-1}\text{ Exit} \rightarrow \bigcirc^{-1}\text{SA})$	469.5 s	WA $\rightarrow$ SA $\rightarrow$ Exit
Visit area 1, then area 2, then area 3	$(\Diamond^{-1}G3 \rightarrow \bigcirc^{-1}G2) \wedge (\Diamond^{-1}G2 \rightarrow \bigcirc^{-1}G1)$	989.7 s	G1 $\rightarrow$ G2 $\rightarrow$ G3
Avoid obstacle areas	$(\Diamond^{-1}G2 \rightarrow \bigcirc^{-1}G1)$	466.1 s	G1 $\rightarrow$ G2

Table 5.4: For each scenario, we obtain the most likely specification using their likelihood estimates and prior probabilities. Goal sequences are derived from the specification.

obtained was exactly or similar in meaning to the specification we expected to infer. However, note also that despite not being the most likely specification, the *avoid* specification did belong to the final set of likely specifications.

In the next section, we present how the goal sequences obtained are leveraged to set prior goal probabilities in a human motion prediction framework with the goal of improving the posterior belief over the human's goals and future states.

5.6 Motion Prediction Framework

Previous work in cognitive science [106], [107], [108], has proposed a model for human motion based on the concept of utility-driven optimization. This concept enables us to think of the human's behavior as being driven by a reward function that depends on the human's state, action and current goal (i.e. with high reward given to the shortest path to her goal). As seen in the IRL framework, this reward function can be computed as a linear combination of features, which capture the preferences of the human over her possible destinations. In this work, we assume that these preferences are set according to the goal sequences derived from the inference results. By applying the principle of maximum entropy [99] to model the human as more likely to choose an action u_H from a finite set of actions $\mathcal{U}$ with high expected utility (or state-action value Q), the probability distribution over the human's actions conditioned on her state is computed as,

$$P(u_H|x_H;\beta;g_H) = \frac{e^{\beta Q_H(x_H,u_H,g_H)}}{\sum_{u\in\mathcal{U}} e^{\beta Q_H(x_H,u,g_H)}}, \tag{5.34}$$

where goal $g_H \in \mathbb{R}^2$, reward function $r_H(x_H,u_H,g_H) = -v_H\Delta t$ and state-action value $Q_H(x_H,u_H,g_H) = -v_H\Delta t - \parallel x_H + v_H[cosu_H, sinu_H]^T\Delta t - g_H \parallel_2$. The first term in Q_H corresponds to the distance traveled over time step Δt or current reward while the second term corresponds to the future reward with respect to how much closer the human will be to her goal after Δt. The parameter β is termed as the *rationality coefficient* in the cognitive science model and is used to quantify the degree to which we expect the human's control choice aligns with its model of utility [106]. As β decreases to 0, the human is modeled to be more "irrational" and expected to choose actions at random. On contrast, as β increases, the human is modeled to be more "rational" and is expected to choose actions that optimize the reward function. Thus, the human is assumed to choose control actions in a Markovian fashion according to the probability distribution shown in Eq. 5.34.

Using the model in Eq. 5.34, we maintain a Bayesian belief about the possible values of g_H. Initially, we begin by setting priors for each known possible goal g_H and update this distribution in real-time given measurements of the human's states and actions. We start with a prior belief b^0_- over the initial value of g_H and update the belief over g_H, $b^i_-(g_H)$, at each discrete time step $i \in \{0, 1, ..., n\}$ by applying Bayes' rule. We denote the updated belief as b^i_+. Since g_H may change over the prediction horizon and we do not have a transition model for g_H, we use a naive transition model. At each time step i, g_H may change with some probability ϵ and be re-sampled from the initial distribution b^0_- or it may stay the same otherwise. By changing ϵ, we can effectively change the rate at which b^i_- approaches b^0_- if no new measurements are obtained. Thus, the belief over the next value of g_H is computed as,

$$b^i_-(g'_H) = (1 - \epsilon) b^{i-1}_+(g'_H) + \epsilon b^0_-(g'_H), \tag{5.35}$$

and the updated belief b^k_+ about g_H is computed as,

$$b^i_+(g_H) = \frac{P(u^i_H | x^i_H, \beta, g_H) b^i_-(g_H)}{\sum_{g \in \mathcal{G}} P(u^i_H | x^i_H, \beta, g) b^i_-(g)}, \tag{5.36}$$

Using the Kolmogorov forward equations, we can propagate the human's state distribution to any future time step $i+1$ as follows,

$$P(x_H^{i+1}, \beta, g_H) = \sum_{x_H^i, u_H^i} P(x_H^{i+1}, u_H^k, \beta, g_H). \\ P(u_H^i | x_H^i, \beta, g_H) P(x_H^i, \beta, g_H), \tag{5.37}$$

for a particular choice of g_H. To obtain the overall occupancy probability distribution at each time step i, we marginalize over g_H as:

$$P(x_H^i, \beta) = \mathbb{E}_{g_H \sim b^i} P(x_H^i, \beta, g_H). \tag{5.38}$$

Since the randomness in f_H comes from the human's choice of control input u_H, we have the following distribution over the human's future states,

$$P(x_H^{i+1} | x_H^i, u_H^i, \beta, g_H) = \mathbb{1}\{x_H^{i+1} = f_H(x_H^i, u_H^i)\}. \tag{5.39}$$

In the next section, we present results for the posterior beliefs over the goals generated by setting the priors according to the goal sequences derived from the inference process.

5.7 Prediction Results

To test the motion prediction framework, we use the discrete dynamics corresponding to the simple, purely kinematic model of continuous-time human motion as,

$$x_H^{i+1} - x_H^i = x_H(t + \Delta t) - x_H(t). \tag{5.40}$$

The implementation of the motion prediction framework was done in Python. We set the parameters $\beta = 1.0$, $v_H = 1.5m/s$ (i.e. the average human walking speed), $\epsilon = 0.02$, and $\Delta = 0.01$. The human control actions are chosen from $\mathcal{U} = \{0, 15, 30, ..., 345\}$. For each scenario tested, we generated 20 trajectories starting from different initial locations sampled according to a uniform distribution over all possible values of x and y in a workspace of 20×20. The objective of the human is to reach each of the goals in order as defined by the goal sequence of each scenario as shown in Table 5.4.

The results for each scenario are displayed in Figs. 5.15- 5.18. The posterior goal probabilities computed during the length of the human trajectory are shown for two types of goal priors: a uniform prior and a prior set according to the goal sequence derived in Table 5.4. We also include results for the case in which the prior is set according to the inference results but the human unexpectedly visits goal locations out of order with respect to the goal sequence defined.

For the scenario, "wait in line if service area is unavailable", we assume there are 4 possible goal locations. For the case of uniform prior, we set $b_-^0 = [0.25, 0.25, 0.25, 0.25]$ corresponding to possible goal locations, $[g_0, g_1, g_2, g_3]$. For the case of the prior set according to the inferred specification, we set $b_-^0 = [0.9, 0.03, 0.03, 0.03]$, respectively. A few outcomes are observable in Fig. 5.15. First, the posterior probabilities for the human's intended goal given a uniform prior is slightly greater than the posterior probabilities for all other goals. In contrast to this, the posterior probabilities for the human's intended goal given the specification prior are much greater than those of the other goals. Second, for the case in which the human does not abide by the inferred specification and goes to a different goal location to the one expected, the posterior probability for the human's intended goal quickly increases despite being set to a low value initially. For all other scenarios which involve a sequence of goal destinations that the human has to visit, we can make the same observations. The goal that the human pursues has the highest posterior probability among all possible goal destinations with the posterior probabilities obtained using the specification prior being much higher.

In Fig. 5.7, we can observe the differences in the occupancy grids computed by using the uniform prior and the specification prior for the scenario, 'wait in line if service area is unavailable, then get service once service area becomes available, then proceed to exit". For the case of uniform prior, we observe an occupancy grid that contains more future states with low probability values, while the occupancy grid for the case of the specification prior shows fewer locations with low probability values. In addition, most of the probability is concentrated on a few states. Thus, there is more confidence about the future states of the human in the case of using the specification prior.

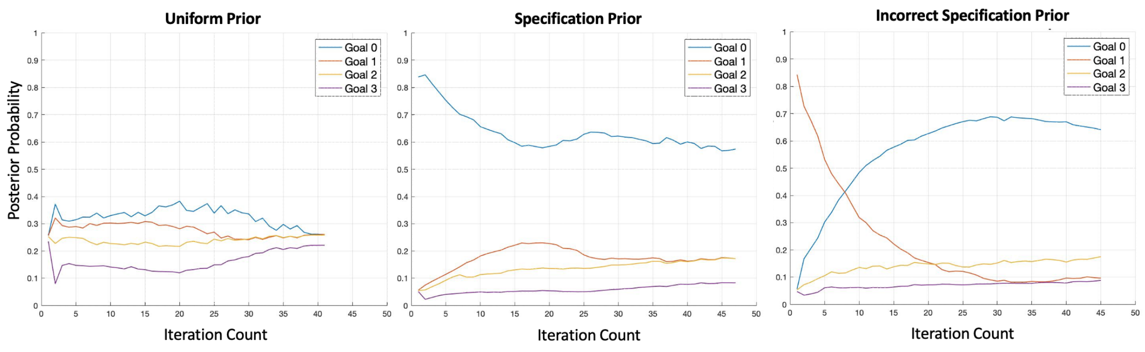


Figure 5.15: Posterior goal probabilities for scenario, "wait in line if service area is unavailable" using a uniform prior (left) and a prior set according to the inference results (middle and right). On the right, we show the case in which the inferred specification is incorrect with respect to the human motion behavior.

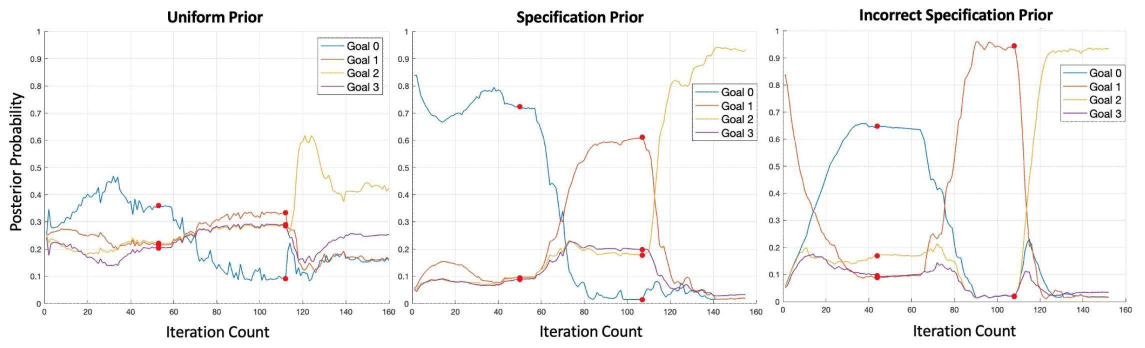


Figure 5.16: Posterior goal probabilities for scenario, "wait in line if service area is unavailable, then get service once service area becomes available, then proceed to exit". Red dots indicate the time steps at which the human reaches a goal and proceeds to the next.

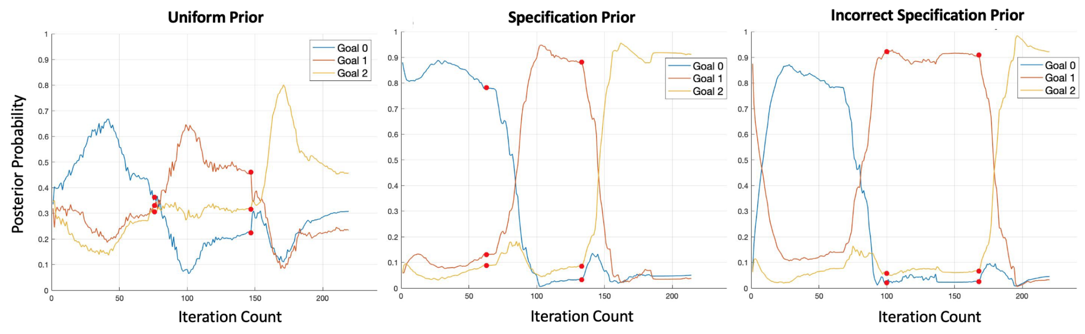


Figure 5.17: Posterior goal probabilities for scenario, "visit area 1, then area 2, then area 3." On the right, we show the case in which the inferred specification is incorrect with respect to the human motion behavior.

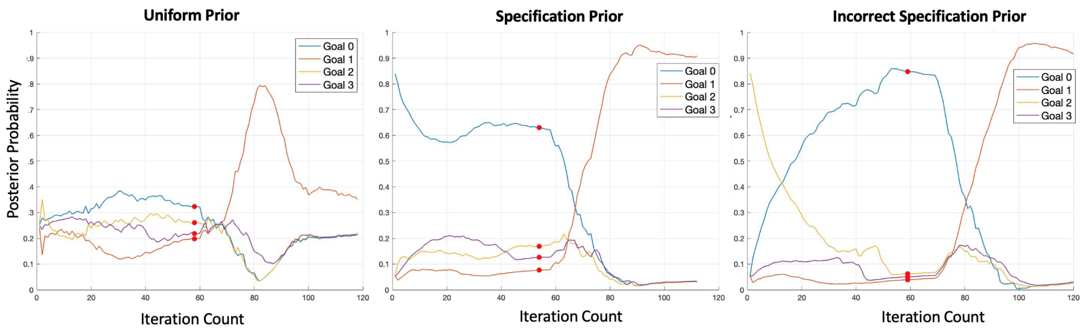
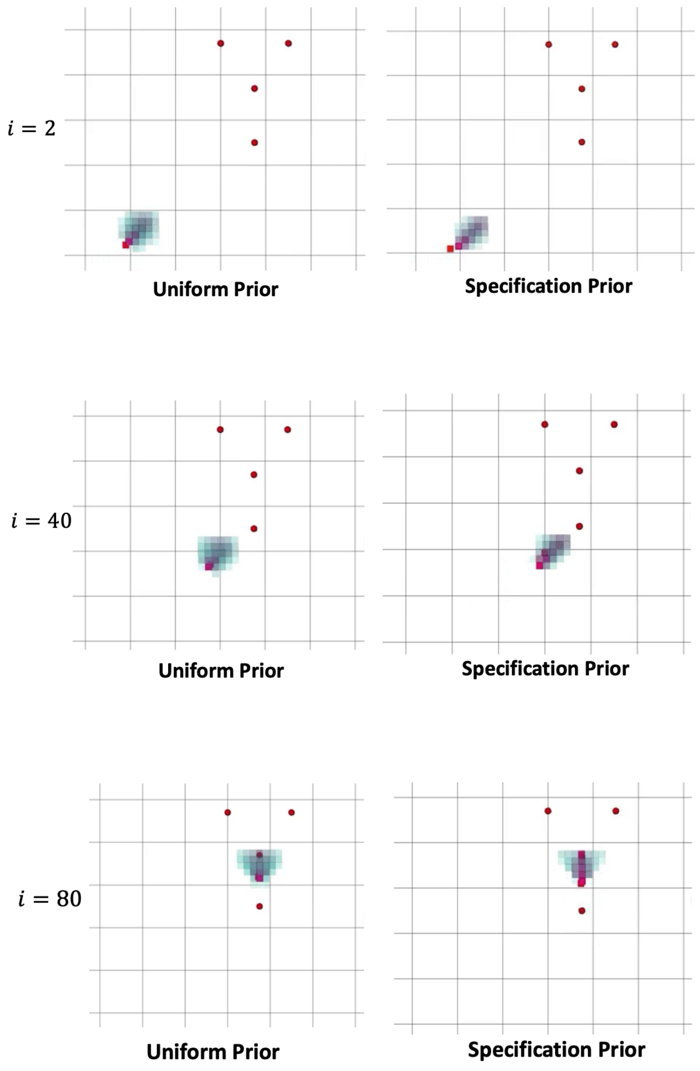


Figure 5.18: Posterior goal probabilities for scenario, "avoid obstacle areas." On the right, we show the case in which the inferred specification is incorrect with respect to the human motion behavior.



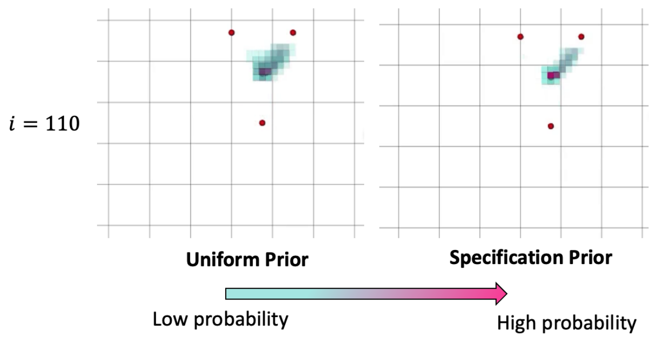


Figure 5.19: Sample occupancy grids generated for scenario, “wait in line if service area is unavailable, then get service once service area becomes available, then proceed to exit”, at different time steps of the simulation. On the left, the uniform prior is used and the occupancy grid appears to be more spread out. On the right, the specification prior is used resulting in a smaller set of future states with higher probability values.

5.8 Conclusions and Future Work

In this chapter, we evaluate and compare three state of the art inference approaches within an Inverse Reinforcement Learning framework to identify the best performing approach. The inference methods are compared in terms of overall runtime including training time, output type, and main drawbacks. Once the best performing method is identified, extensions were proposed to automatize the generation of demonstrations satisfying a given LTL specification. Our contributions include the systematic construction process of an initial hypothesis space of specifications, a process of refinement applied to systematically perturb likely specifications to find more likely and complete specifications that convey the demonstrated behavior, and a human motion prediction framework that uses the inference results to set priors for all possible known goals that the human is expected to visit. Through various scenarios, we demonstrate the effectiveness of the inference process in obtaining the correct LTL specification and enhanced human motion prediction results in terms of larger posterior probabilities for the human's intended goals. The resulting occupancy grids also show a higher confidence on the human's future states.

With respect to the training data generation aspect of this chapter, one possible extension is to use motion capture technology to collect human trajectories. Observing real data can help us address questions on how to identify the set of atomic propositions for a particular environment. With respect to the human motion prediction aspect, future work can be done to adapt the rationality coefficient as well as the belief over the goals in order to improve prediction results at the cost of increasing computational effort. Additionally, this framework can be integrated with a safe-navigation framework that robots can operate within to safely maneuver around humans given their predicted occupancy grids.

Bibliography

[1] M. Cox and A. Raja, *Metareasoning: An Introduction.* The MIT Press,2011, pp. 3–14.

[2] G. Alexander, A. Raja, E. H. Durfee, and D. J. Musliner, "Design Paradigms for Meta-Control in Multiagent Systems," in *Proc. of AAMAS Workshop on Metareasoning in Agent-based Systems*, 2007, pp. 92–103.

[3] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Transactions On Automatic Control*, vol. 57, no. 11, pp. 2817-2830, Nov. 2012.

[4] A. Ulusoy and C. Belta, "Receding horizon temporal logic control in dynamic environments," *The International Journal of Robotics Research*, vol. 33, no. 12, pp. 1593-1607, 2014, doi: 10.1177/0278364914537008.

[5] X. C. Ding and M. Lazar and C. Belta, "Receding horizon temporal logic control for finite deterministic systems," in *Proc. Amer. Control Conf. (ACC)*, Jun. 2012, pp. 715-720.

[6] E. Carrillo, S. Yeotikar, S. Nayak, M.K.M. Jaffar, S. Azarm, M. Otte, and H. Xu, "Communication-aware multi-agent metareasoning for decentralized task allocation," *IEEE Access*, *under review*, April 2021.

[7] J. A. Shaffer, E. Carrillo, and H. Xu, "Hierarchal application of receding horizon synthesis and dynamic allocation for uavs fighting fires," *IEEE Access*, vol. 6, pp. 78 868–78 880, 2018.

[8] M. P. Plucinski *et al.*, "The effectiveness and efficiency of aerial firefighting in Australia. Part 1," Bushfire Cooperat. Res. Centre, Melbourne, VIC, Australia, Tech. Rep. A0701, 2006.

[9] D. Thomas, D. Butry, S. Gilbert, D. Webb, and J. Fung, “The costs and losses of wildfires: A literature survey,” NIST Special Publication, Gaithersburg, MD, USA, Tech. Rep. 1215, Nov. 2017.

[10] D. Werner, “Fire drones,” *Aerosp. Amer.*, vol. 53, no. 6, pp. 28-31, 2015.

[11] L. Merino, F. Caballero, J. R. Martínez-de Dios, I. Maza, and A. Ollero, “An unmanned aircraft system for automatic forest fire monitoring and measurement,” *J. Intell. Robotic Syst.*, vol. 65, nos. 1-4, pp. 533-548, Jan. 2012, doi: 10.1007/s10846-011-9560-x.

[12] L. Martin, *Lockheed Martin Conducts Collaborative Unmanned Systems Demonstration.* [Online]. Available: https://news.lockheedmartin.com/2015-12-02-Lockheed-Martin-Conducts-Collaborative-Unmanned-Systems-Demonstration

[13] H. Qin *et al.*, “Design and implementation of an unmanned aerial vehicle for autonomous firefighting missions,” in *Proc. 12th IEEE Int. Conf. Control Autom. (ICCA)*, Jun. 2016, pp. 62-67.

[14] B. Johnson, F. Havlak, H. Kress-Gazit, and M. Campbell, “Experimental evaluation and formal analysis of high-level tasks with dynamic obstacle anticipation on a full-sized autonomous vehicle,” *J. Field Robot.*, vol. 34, no. 5, pp. 897-911, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21695

[15] K. W. Wong and H. Kress-Gazit, “Need-based coordination for decentralized high-level robot control,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Oct. 2016, pp. 2209-2216.

[16] J. Alonso-Mora, J. A. DeCastro, V. Raman, D. Rus, and H. Kress-Gazit, “Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles,” *Auton. Robots*, vol. 42, no. 4, pp. 801-824, Apr. 2018, doi: 10.1007/s10514-017-9665-6.

[17] B. Mirzapour and H. S. Aghdasi, “Modified multi-objective hybrid DE and PSO algorithms for resource allocation in forest fires,” in *Proc. 7th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct. 2017, pp. 187-192.

[18] J. D. Griffith, M. J. Kochenderfer, R. J. Moss, V. V. Misic, V. Gupta, and D. Bertsimas, "Automated dynamic resource allocation for wildfire suppression," *Lincoln Lab. J.*, vol. 22, no. 2, pp. 38-59, 2017.

[19] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239-248, Aug. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0005109816301285

[20] N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of reactive(1) designs," in *Verification, Model Checking, and Abstract Interpretation*, E. A. Emerson and K. S. Namjoshi, Eds. Berlin, Germany: Springer, 2006, pp. 364-380.

[21] M. A. Finney, "FARSITE: Fire area simulator–Model development and evaluation," U.S. Dept. Agricult., Forest Service, Rocky Mountain Res. Station, Ogden, UT, USA, Res. Paper RMRS-RP-4, 2004, p. 47.

[22] P. L. Andrews, "The Rothermel surface fire spread model and associated developments: A comprehensive explanation," U.S. Dept. Agricult., Forest Service, Rocky Mountain Res. Station, Fort Collins, CO, USA, Tech. Rep. RMRS-GTR-371, Mar. 2018.

[23] T.Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "TuLiP: A software toolbox for receding horizon temporal logic planning," in *Proc. 14th Int. Conf. Hybrid Syst., Comput. Control,* New York, NY, USA, 2011, pp. 313-314, doi: 10.1145/1967701.1967747.

[24] J. S. Gould et al., "Assessment of the effectiveness and environmental risk of the use of retardants to assist in wildfire control in Victoria," Austral. Dept. Natural Resour. Environ., CSIRO, Canberra, SW, Australia, Tech. Rep. 50, Feb. 2000.

[25] J. Sethian, "A fast marching level set method for monotonically advancing fronts," *National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[26] S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces.* Springer-Verlag, 2006.

[27] I. Mitchell, "Application of level set methods to control and reachability problems in continuous and hybrid systems," Ph.D. dissertation, Stanford University, 2002.

[28] I. Mitchell *et al.*, "A toolbox of level set methods," Department of Computer Science, University of British Columbia, Vancouver, BC, Canada, http://www.cs.ubc.ca/˜mitchell/ToolboxLS/toolboxLS.pdf, Tech. Rep. TR-2004-09, 2004.

[29] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal and C. J. Tomlin, "Decomposition of Reachable Sets and Tubes for a Class of Nonlinear Systems," in *IEEE Transactions on Automatic Control*, vol. 63, no. 11, pp. 3675-3688, Nov. 2018, doi: 10.1109/TAC.2018.2797194.

[30] A. Majumdar, A. Ahmadi, and R. Tedrake, "Control design along trajectories with sums of squares programming," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2013.

[31] M. Otte, M. J. Kuhlman, and D. Sofge, "Auctions for multirobot task allocation in communication limited environments," *Autonomous Robots*, Jan 2019. [Online]. Available: https://doi.org/10.1007/s10514-019-09828-5

[32] S. Nayak, S. Yeotikar, E. Carrillo, E. Rudnick-Cohen, M. K. M. Jaffar, R. Patel, S. Azarm, J. W. Herrmann, H. Xu, and M. Otte, "Experimental Comparison of Decentralized Task Allocation Algorithms Under Imperfect Communication," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 572–579, April 2020. [Online]. Available: https://ieeexplore.ieee.org/document/8949727

[33] A. Muralidharan and Y. Mostofi, "Communication-Aware Robotics: Exploiting Motion for Communication," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, May 2021, doi: 10.1146/annurev-control-071420-080708, [Online].

[34] M. Rantanen, N. Mastronarde, J. Hudack and K. Dantu, "Decentralized Task Allocation in Lossy Networks: A Simulation Study," *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, Boston, MA, USA, 2019, pp. 1-9, doi: 10.1109/SAHCN.2019.8824898

[35] M. Nanjanath and M. Gini, "Repeated auctions for robust task execution by a robot team," *Robotics and Autonomous Systems*, vol. 58, no. 7, pp. 900 – 909, 2010, advances in Autonomous Robots for Service and Entertainment. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889010000692

[36] J. Wang, Y. Gu, and X. Li, “Multi-robot task allocation based on ant colony algorithm,” *Journal of Computers*, vol. 7, no. 9, pp. 2160–2167, 2012.

[37] C. Sabo, D. Kingston and K. Cohen, “A Formulation and Heuristic Approach to Task Allocation and Routing of UAVs under Limited Communication,” *Unmanned Systems*, vol. 2, no. 1, pp. 1-17, January 2014, doi: 10.1142/S2301385014500010

[38] A. Raja and V. Lesser, “A framework for meta-level control in multi-agent systems,” *Autonomous Agents and Multi-Agent Systems*,vol. 15, no. 2, pp. 147–196, Oct 2007. [Online]. Available: https://doi.org/10.1007/s10458-006-9008-z

[39] S. Cheng, A. Raja, and L. Victor, “Multiagent Meta-level Control for Radar Coordination,” *Web Intelligence and Agent Systems: An International Journal*, vol. 11, no. 2, pp. 81–105, 2013. [Online]. Available: http://mas.cs.umass.edu/paper/512

[40] R. Becker, S. Zilberstein, V. Lesser, and C. Goldman, “Solving transition independent decentralized markov decision processes,” *J. Artif. Intell. Res. (JAIR)*, vol. 22, pp. 423–455, 07 2004.

[41] H.-L. Choi, L. Brunet, and J. P. How, “Consensus-based decentralized auctions for robust task allocation,” *IEEE transactions on robotics*, vol. 25, no. 4, pp. 912–926, 2009.

[42] L. B. Johnson, S. S. Ponda, H.-L. Choi, and J. P. How, “Asynchronous Decentralized Task Allocation for Dynamic Environments,” in *AIAA Infotech@Aerospace Conference*. AIAA, 2011. [Online]. Available: http://hdl.handle.net/1721.1/81434

[43] L. Johnson, S. Ponda, H.-L. Choi, and J. How, “Improving the efficiency of a decentralized tasking algorithm for uav teams with asynchronous communications,” in *AIAA Guidance, Navigation, and Control Conference*, 2010, p. 8421.

[44] S. Ismail and L. Sun, “Decentralized hungarian-based approach for fast and scalable task allocation,” in *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2017, pp. 23–28. [Online]. Available: https://ieeexplore.ieee.org/document/7991447

[45] L. Johnson, H.-L. Choi, and J. P. How, “The hybrid information and plan consensus algorithm with imperfect situational awareness,” in *Distributed Autonomous Robotic Systems*. Springer, 2016, pp. 221– 233.

[46] L. B. Johnson, H.-L. Choi, and J. P. How, “Hybrid information and plan consensus in distributed task allocation,” in *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2013, p. 4888.

[47] W. Zhao, Q. Meng, and P. W. Chung, “A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario,” *IEEE transactions on cybernetics*, vol. 46, no. 4, pp. 902–915, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7084641

[48] Y. R. Zheng and C. Xiao, “Simulation models with correct statistical properties for Rayleigh fading channels,” *IEEE Transactions on Communications*, vol. 51, no. 6, pp. 920–928, 2003. [Online]. Available: https://ieeexplore.ieee.org/document/1209292

[49] M. T. Cox, “Metacognition in Computation: A Selected Research Review,” *Artif. Intell.*, vol. 169, no. 2, pp. 104–141, Dec. 2005. [Online]. Available: https://doi.org/10.1016/j.artint.2005.10.009

[50] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, “Automated Algorithm Selection: Survey and Perspectives,” *CoRR*, 2018. [Online]. Available: http://arxiv.org/abs/1811.11597

[51] M. Cox and A. Raja, *Metareasoning: An Introduction*. The MIT Press, 2011, pp. 3–14.

[52] J. Sleight and E. H. Durfee, “Multiagent Metareasoning Through Organizational Design,” in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, ser. AAAI’14. AAAI Press, 2014, pp. 1478–1484. [Online]. Available: https://dl.acm.org/doi/10. 5555/2892753.2892758

[53] S. Karaman and E. Frazzoli, “Vehicle routing with linear temporal logic specifications: Applications to multi-uav mission planning,” *International Journal of Robust and Nonlinear Control*, vol. 21, pp. 1372 – 1395, 08 2011.

[54] M. S. Andersen, R. S. Jensen, T. Bak, and M. M. Quottrup, “Motion planning in multi-robot systems using timed automata,” *IFAC Proceedings Volumes*,

vol. 37, no. 8, pp. 597 – 602, 2004, iFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004. [Online]. Available: http://www.sciencedirect. com/science/article/pii/S1474667017320438

[55] H. Kress-Gazit, D. C. Conner, H. Choset, A. A. Rizzi, and G. J. Pappas, "Courteous cars," *IEEE Robotics Automation Magazine*, vol. 15, no. 1, pp. 30–38, 2008.

[56] D. Dimarogonas and K. Kyriakopoulos, "Decentralized navigation functions for multiple robotic agents with limited sensing capabilities," *Journal of Intelligent and Robotic Systems*, vol. 48, pp. 411–433, 02 2007.

[57] C. K. Verginis and D. V. Dimarogonas, "Motion and cooperative transportation planning for multi-agent systems under temporal logic formulas," *ArXiv*, vol. abs/1803.01579, 2018.

[58] H. G. Tanner, S. G. Loizou, and K. J. Kyriakopoulos, "Nonholonomic navigation and control of cooperating mobile manipulators," IEEE Transactions on Robotics and Automation, vol. 19, no. 1, pp. 53–64, 2003.

[59] Y. Chen, X. Ding, A. Stefanescu, and C. Belta, "Formal approach to the deployment of distributed robotic teams," *IEEE Transactions on Robotics*, vol. 28, pp. 158–171, 02 2012.

[60] J. Liu, N. Ozay, U. Topcu, and R. M. Murray, "Synthesis of Reactive Switching Protocols From Temporal Logic Specifications," *IEEE Transactions on Automatic Control*, vol. 58, no. 7, pp. 1771–1785, 2013. [Online]. Available: https://ieeexplore.ieee.org/document/6457409

[61] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-Logic-Based Reactive Mission and Motion Planning," IEEE Transactions on Robotics, vol. 25, no. 6, pp. 1370–1381, 2009. [Online]. Available: https://ieeexplore.ieee.org/document/5238617

[62] T. J. Koo, G. J. Pappas, and S. Sastry, "Mode Switching Synthesis for Reachability Specifications," in *Hybrid Systems: Computation and Control*, M. D. Di Benedetto and A. Sangiovanni-Vincentelli, Eds. Springer Berlin Heidelberg, 2001, pp. 333–346.

[63] B. Johnson, F. Havlak, H. Kress-Gazit, and M. Campbell, "Experimental evaluation and formal analysis of high-level tasks with dynamic obstacle anticipation on a full-sized autonomous vehicle," *J. Field Robot.*, vol. 34, no. 5, pp. 897-911, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21695

[64] P. Nilsson and N. Ozay, "Incremental synthesis of switching protocols via abstraction refinement," in *53rd IEEE Conference on Decision and Control*, Dec 2014, pp. 6246–6253.

[65] H. Xu, U. Topcu, and R. M. Murray, "Specification and synthesis of reactive protocols for aircraft electric power distribution," *IEEE Transactions on Control of Network Systems*, vol. 2, no. 2, pp. 193–203, 2015. [Online]. Available: https://ieeexplore.ieee.org/document/7035090

[66] M. Nanjanath and M. Gini, "Repeated auctions for robust task execution by a robot team," *Robotics and Autonomous Systems*, vol. 58, no. 7, pp. 900 – 909, 2010, advances in Autonomous Robots for Service and Entertainment. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889010000692

[67] J. Wang, Y. Gu, and X. Li, "Multi-robot task allocation based on ant colony algorithm," *Journal of Computers*, vol. 7, no. 9, pp. 2160–2167, 2012.

[68] P. Ghassemi and S. Chowdhury, "Decentralized Task Allocation in Multi-Robot Systems via Bipartite Graph Matching Augmented With Fuzzy Clustering", ser. *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. Volume 2A: 44th Design Automation Conference, 08 2018.

[69] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[70] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations research*, vol. 35, no. 2, pp. 254–265, 1987.

[71] N. Ozay, J. Liu, P. Prabhakar, and R. M. Murray, "Computing augmented finite transition systems to synthesize switching protocols for polynomial switched systems," in *2013 American Control Conference*, 2013, pp. 6237–6244. [Online]. Available: https://ieeexplore.ieee.org/ document/6580816

[72] R. Patel, "Multi-Vehicle Route Planning for Centralized and Decentralized Systems," Master's thesis, University of Maryland, 2019. [Online]. Available: http://hdl.handle.net/1903/25197

[73] H. Choi, Y. Kim, and H. Kim, "Genetic Algorithm Based Decentralized Task Assignment for Multiple Unmanned Aerial Vehicles in Dynamic Environments," International Journal of Aeronautical and Space Sciences, vol. 12, no. 2, pp. 163–174, 2011.

[74] B. P. Gerkey and M. J. Mataric, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004. [Online]. Available: https://doi.org/10.1177/0278364904045564

[75] X. Jia and M. Q.-H. Meng, "A survey and analysis of task allocation algorithms in multi-robot systems," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 2013, pp. 2280– 2285.

[76] G. Alexander, A. Raja, E. H. Durfee, and D. J. Musliner, "Design Paradigms for Meta-Control in Multiagent Systems," in *Proc. of AAMAS Workshop on Metareasoning in Agent-based Systems*, 2007, pp. 92–103.

[77] L. Brunet, H.-L. Choi, and J. How, "Consensus-Based Auction Approaches for Decentralized Task Assignment," in *AIAA Guidance, Navigation and Control Conference and Exhibit*, 2008, p. 6839. [Online]. Available: https://doi.org/10.2514/6.2008-6839

[78] H. W. Kuhn, The Hungarian Method for the Assignment Problem. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 29–47. [Online]. Available: https://doi.org/10.1007/978-3-540-68279-0 2

[79] J. I. Smith, "A computer generated multipath fading simulation for mobile radio," *IEEE Transactions on Vehicular Technology*, vol. 24, no. 3, pp. 39–40, 1975. [Online]. Available: https: //ieeexplore.ieee.org/document/1622250

[80] D. J. Young and N. C. Beaulieu, "The generation of correlated Rayleigh random variates by inverse discrete Fourier transform," *IEEE Transactions on Communications*, vol. 48, no. 7, pp. 1114–1127, 2000. [Online]. Available: https://ieeexplore.ieee.org/document/855519

[81] A. Pnueli, Y. Saar, and L. D. Zuck, "JTLV: A framework for developing verification algorithms," in *Computer Aided Verification (Lecture Notes in Computer Science)*, T. Touili, B. Cook, and P. Jackson, Eds. Berlin, Germany: Springer-Verlag, 2010, pp. 171-174.

[82] A. Pnueli and R. Rosner, "Distributed reactive systems are hard to synthesize," in *Proc. 31st Annu. Symp. Found. Comput. Sci.*, vol. 2. Oct. 1990, pp. 746-757.

[83] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar, "Synthesis of Reactive(1) designs," *Journal of Computer and System Sciences*, vol. 78, no. 3, pp. 911–938, 2012. [Online]. Available: https://doi.org/10.1016/j.jcss.2011.08.007

[84] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 2, 2002, pp. 1327–1332 vol.2.

[85] S. He, J. Chen, and Y. Sun, "Coverage and connectivity in duty-cycled wireless sensor networks for event monitoring," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 3, pp. 475–482, 2012.

[86] K. S. V. Narasimha and M. Kumar, "Ant colony optimization technique to solve the min-max single depot vehicle routing problem," in *Proceedings of the 2011 American Control Conference*. IEEE, 2011, pp. 3257–3262.

[87] M. Yong, M. Koes, K. Sycara, M. Lewis, I. R. Nourbakhsh, and S. Burion, "Human-robot teaming for search and rescue," *IEEE Pervasive Computing*, vol. 4, no. 01, pp. 72–78, jan 2005.

[88] A. Koubaa, *Robot Operating System (ROS): The Complete Reference (Volume 2)*, 1st ed. Springer Publishing Company, Incorporated, 2017. [Online]. Available: https://www.springer.com/gp/book/9783319549262

[89] T. S. Rappaport et al., *Wireless communications: principles and practice*. Prentice hall PTR New Jersey, 1996, vol. 2.

[90] M. D. McKay, R. J. Beckman, and W. J. Conover, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," Technometrics, vol. 21, no. 2, pp. 239–245, 1979.

[91] F. Wilcoxon, “Individual comparisons by ranking methods,” Biometrics Bulletin, vol. 1, no. 6, pp. 80–83, 1945.

[92] S. T. Langlois, O. Akoroda, E. Carrillo, S. Azarm, M. Otte, H. Xu and J. W. Herrmann, “Metareasoning Structures, Problems, and Modes for Multiagent Systems: A Survey,” in *IEEE Access*, vol. 8, pp. 183080-183089, 2020, doi:10.1109/ACCESS.2020.3028751

[93] A. Y. Ng and S. J. Russell, “Algorithms for Inverse Reinforcement Learning,” in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML ’00)*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 663–670.

[94] R.T. Icarte, T. Klassen, R. Valenzano, and McIlraith S., “Using Reward Machines for High-Level Task Specification and Decomposition in Reinforcement Learning,” in *Proceedings of the 35th International Conference on Machine Learning*, pp. 2112-2121, 2018.

[95] K. Jothimurugan, R. Alur, and O. Bastani, “A composable specification language for reinforcement learning tasks,” in *Advances in Neural Information Processing Systems*, pp. 13021-13030, 2019.

[96] D. Kasenberg and M. Scheutz, “Interpretable apprenticeship learning with temporal logic specifications,” *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, 2017, pp. 4914-4921.

[97] B. Farwer, w-automata. In: E. Gradel, W. Thomas, T. Wilke (eds.) Automata Logics and Infinite Games. LNCS, vol. 2500, pp. 3-21. Springer, Heidelberg (2002).

[98] K. Shiarlis, J. Messias, and S. Whiteson, “Inverse Reinforcement Learning from Failure,” in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems (AAMAS ’16) International Foundation for Autonomous Agents and Multiagent Systems*, Richland, SC, pp. 1060–1068, 2016.

[99] B. D. Ziebart, J. A. Bagnell, and A. K. Dey, “Modeling interaction via the principle of maximum causal entropy,” in *ICML*, 2010, pp. 1255–1262.

[100] M. Vazquez-Chanlatte and S. A. Seshia, “Maximum causal entropy specification inference from demonstrations,” in *Computer Aided Verification*, S.

K. Lahiri and C. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 255–278

[101] H. Kamp, “Tense Logic and the Theory of Linear Order,” Dissertation, UCLA, 1968.

[102] P. Trautman and A. Krause, “Unfreezing the robot: Navigation in dense,interacting crowds,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 797–803.

[103] A. Lavaei, F. Somenzi, S. Soudjani, A. Trivedi, and M. Zamani, “Formal controller synthesis for continuous-space mdps via model-free reinforcement learning,” 2020, pp. 98–107.

[104] O. Kupferman and M. Vardi, “Model checking of safety properties,” Formal Methods in System Design, vol. 19, 09 1999.

[105] A. Lavaei, M. Khaled, S. Soudjani, and M. Zamani, “AMYTISS: PArallel AutoMated Controller SYnthesis for Large-Scale STochastIc SystemS”, in *32nd International Conference on Computer-Aided Verification (CAV)*, Lecture Notes in Computer Science, Springer, to appear, 2020.

[106] J. V. Neumann, O. Morgenstern, and H. W. Kuhn. *Theory of games and economic behavior (commemorative edition)*. Princeton university press, 2007. ISBN: 0691130612.

[107] R. D. Luce. *Individual choice behavior: A theoretical analysis.* Courier Corporation, 2012. ISBN: 0486153398.

[108] C. L. Baker, J. B. Tenenbaum, and R. R. Saxe, “Goal inference as inverse planning,” in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 29, 2007.