# ABSTRACT

Title of Dissertation: ASSURED AUTONOMY IN MULTIAGENT SYSTEMS WITH SAFE LEARNING

Usman Amin Fiaz
Doctor of Philosophy, 2022

Dissertation Directed by: Professor John S. Baras
Department of Electrical & Computer Engineering

Autonomous multiagent systems is an area that is currently receiving increasing attention in the communities of robotics, control systems, and machine learning (ML) and artificial intelligence (AI). It is evident today, how autonomous robots and vehicles can help us shape our future. Teams of robots are being used to help identify and rescue survivors in case of a natural disaster for instance. There we understand that we are talking minutes and seconds that can decide whether you can save a person's life or not. This example portrays not only the value of safety but also the significance of time, in planning complex missions with autonomous agents.

This thesis aims to develop a generic, composable framework for a multiagent system (of robots or vehicles), which can safely carry out time-critical missions in a distributed and autonomous fashion. The goal is to provide formal guarantees on both safety and finite-time mission completion in real time, thus, to answer the question: "how trustworthy is the autonomy of a multi-robot system in a complex mission?" We refer to this notion of autonomy in multiagent

systems as assured or trusted autonomy, which is currently a very sought-after area of research, thanks to its enormous applications in autonomous driving for instance.

There are two interconnected components of this thesis. In the first part, using tools from control theory (optimal control), formal methods (temporal logic and hybrid automata), and optimization (mixed-integer programming), we propose multiple variants of (almost) realtime planning algorithms, which provide formal guarantees on safety and finite-time mission complet--ion for a multiagent system in a complex mission. Our proposed framework is hybrid, distributed, and inherently composable, as it uses a divide-and-conquer approach for planning a complex mission, by breaking it down into several sub-tasks. This approach enables us to implement the resulting algorithms on robots with limited computational power, while still achieving close to realtime performance. We validate the efficacy of our methods on multiple use cases such as autonomous search and rescue with a team of unmanned aerial vehicles (UAVs) and ground robots, autonomous aerial grasping and navigation, UAV-based surveillance, and UAV-based inspection tasks in industrial environments.

In the second part, our goal is to translate and adapt these developed algorithms to safely learn actions and policies for robots in dynamic environments, so that they can accomplish their mission even in the presence of uncertainty. To accomplish this goal, we introduce the ideas of self-monitoring and self-correction for agents using hybrid automata theory and model predictive control (MPC). Self-monitoring and self-correction refer to the problems in autonomy where the autonomous agents monitor their performance, detect deviations from normal or expected behavior, and learn to adjust both the description of their mission/task and their performance online, to maintain the expected behavior and performance. In this setting, we propose a formal and composable notion of safety and adaptation for autonomous multiagent systems, which we

refer to as safe learning. We revisit one of the earlier use cases to demonstrate the capabilities of our approach for a team of autonomous UAVs in a surveillance and search and rescue mission scenario.

Despite portraying results mainly for UAVs in this thesis, we argue that the proposed planning framework is transferable to any team of autonomous agents, under some realistic assumptions. We hope that this research will serve several modern applications of public interest, such as autopilots and flight controllers, autonomous driving systems (ADS), autonomous UAV missions such as aerial grasping and package delivery with drones etc., by improving upon the existing safety of their autonomous operation.

# ASSURED AUTONOMY IN MULTIAGENT SYSTEMS WITH SAFE LEARNING

by

Usman Amin Fiaz

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
2022

Advisory Committee:
Professor John S. Baras, Chair/Advisor
Professor Dinesh Manocha
Professor Eyad H. Abed
Professor Huan (Mumu) Xu
Professor Nikhil Chopra

© Copyright by
Usman Amin Fiaz
2022

*To my family*

# Acknowledgments

I want to thank my PhD advisor, Prof. John Baras, for his gracious advice and continuous encouragement throughout my PhD. He has been an amazing source of inspiration, who I will always look up to as a role model in my academic career. I cannot imagine to have gone through my graduate studies without his generous support. I also want to thank my Master's advisor Prof. Jeff Shamma, for giving me my first shot at research, and for introducing me to the world of control theory and multiagent systems. There is a rare sophistication and unique finesse to his work, and I am very fortunate to have been able to collaborate with and learn from him all these years. Both John and Jeff have been incredible mentors and the best advisors I could have ever asked for. They have an undeniable impact on shaping me into the person I am today, and how I view science and research in general, and for that I am eternally grateful.

I would also like to thank the members of my thesis committee; Prof. Dinesh Manocha, Prof. Eyad Abed, Prof. Mumu Xu, and Prof. Nikhil Chopra. Whether through teaching courses, random discussions, class projects, or formal collaborations, all of these amazing professors have been instrumental in making my time at UMD rewarding. I also had the pleasure of working with Prof. Dinesh during my graduate studies, on problems that eventually became part of this thesis, for which I am grateful. I also want to thank Prof. Pratap Tokekar for his advice and insightful comments on my research. I also want to say thanks to Prof. Gilmer Blankenship for his mentorship and advice early in my graduate student career. There are several other

members of the UMD faculty, who have contributed in one or more ways, to make my time here worthwhile. I want to thank all of them because without their guidance and support, this thesis would almost certainly look very different if not entirely impossible. I would also like to acknowledge all my internship hosts, mentors, and colleagues from Industry, who over all these years, have provided me with the best environment to enhance my knowledge and work on many interesting yet challenging problems in the field. I also want to thank Ms. Melanie Prange for her sincere concern and incredible support during the early years of my graduate studies. Also a huge shoutout to Mrs. Kimberly Edwards for everything she does for our group on the administrative end, just so we can solely focus on our research. I also want to acknowledge the Office of Naval Research (ONR) for the gracious financial support that they provided throughout my PhD, via the grant no. N00014-17-1-2622.

I would be certainly at fault not to acknowledge my friends, colleagues and labmates: the people of ARC and SEIL, who have truly made my time at UMD delightful. Special mentions for my old labmates: Dr. Dipankar, Dr. Aneesh, Dr. Zhenyu, Dr. Leda, and Dr. Christos, and my current office mates: Dr. Fatemeh and (soon to be Dr.) Anousheh, for making our daily conversations constructive, destructive, and fun, all at the same time. I must also thank my friends Dr. Waseem, and Dr. Daanish, for making sure to drag me out of my office from time to time, to have some fun. I cannot emphasize enough how much your friendship has meant to me over the past few years, and I cannot imagine having survived my time here without you, especially during/after the pandemic.

Of course, I never would have made it to or through graduate school without the immeasur--able love and support of my mother Nusrat, father Muhammad, brother Salman, and sister Samia. I dedicate this thesis to them as a symbol of my appreciation for all that they have done for me.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1: Introduction

## 1.1 Background

The primary motivation for this thesis comes from the increasing zeal of modern industry in employing autonomous robots for complex, and safety and time-critical missions. Robots are known to outperform humans when it comes to certain difficult and repetitive tasks. These include but are not limited to supporting search and rescue missions [1, 2], surveillance [3], and exploration [4], periodic inspection of safety-critical equipment and infrastructure [5], and numerous civil applications of unmanned aerial vehicles (UAVs) [6], such as aerial grasping [7] and transport of packages [8], and programmable self-assembly for construction of modular structures [9, 10], for instance. Figure 1.1 shows some representative applications where autono--mous robots have been shown to make a tremendous impact on society.

Often multiple robots are used for these missions to improve both the time efficiency and


Figure 1.1: Some representative applications of autonomous robots where they can be potent in a multiagent setting. (Photo credits: DARPA and NASA)

the cost of execution. This is where considering an autonomous team of robots as a multiagent system comes in quite handy. This way, we can conveniently use the extensive developments from existing literature and the vast amounts of ongoing research on planning missions (which may include multiple tasks) for multiagent systems, to provide certain guarantees on the behavior of the robots involved.

This work builds upon the same line of thought by providing a compositional and hybrid approach to planning and executing safety and time-critical missions with either a single or multiple number of autonomous agents (or robots) in a complex environment. The primary emphasis here is on providing assurances on safety as well as finite-time completion of a given mission. In addition, a secondary goal is to maintain as close as possible to a realtime performance capability. Combining these two objectives together in a systematic and scalable way is what makes this thesis's contributions both novel and unique.

Unless specified otherwise, throughout this thesis, we primarily use UAVs as a default model for our multiagent system. That is because multirotor UAVs are highly inexpensive robots which can be extensively used as testbeds for much of the ongoing research in multiagent robotics. In addition, these are extremely agile aircraft, capable of much higher maneuverability in comparison with the other UAV classes, namely fixed-wing and helicopter style UAVs. This salient feature also edges them as a feasible platform to operate in congested environments, such as crowded city skies and constrained indoor workspaces.

Despite this heavy lean towards UAVs in this thesis, the planning framework we present here is quite generic and is applicable to a number of other multiagent systems as well under some realistic assumptions. We elaborate this idea further towards the end of this thesis.

## 1.2 Main Contributions

This thesis aims to develop a generic, composable framework for a multiagent system (of robots or vehicles), which can safely carry out time-critical missions in a distributed and autonomous fashion. The goal is to provide formal guarantees on both safety and finite-time mission completion in real time, thus, to answer the question: "how trustworthy is the autonomy of a multi-robot system in a complex mission?" We refer to this notion of autonomy in multiagent systems as assured or trusted autonomy, which is currently a very sought-after area of research, thanks to its enormous applications in autonomous driving for instance.

There are two interconnected components of this thesis. In the first part, using tools from control theory (optimal control), formal methods (temporal logic and hybrid automata), and optimization (mixed-integer programming), we propose three variants of (almost) realtime planning algorithms, which provide formal guarantees on safety and finite-time mission complet--ion for a multiagent system in a complex mission. Our proposed framework is hybrid and inherently composable, as it uses a divide-and-conquer approach for planning a complex mission, by breaking it down into several sub-tasks. This approach enables us to implement the resulting algorithms on robots with limited computational power, while still achieving close to realtime performance. We validate the efficacy of our method on three use cases namely an autonomous search and rescue mission with a team of UAVs as well as with a team of ground robots in a leader-follower setting, and a UAV-based inspection task in a smart factory environment.

In the second part, our goal is to translate and adapt these developed algorithms to safely learn actions and policies for robots in dynamic environments, so that they can accomplish their mission even in the presence of uncertainty. To accomplish this goal, we introduce the ideas of

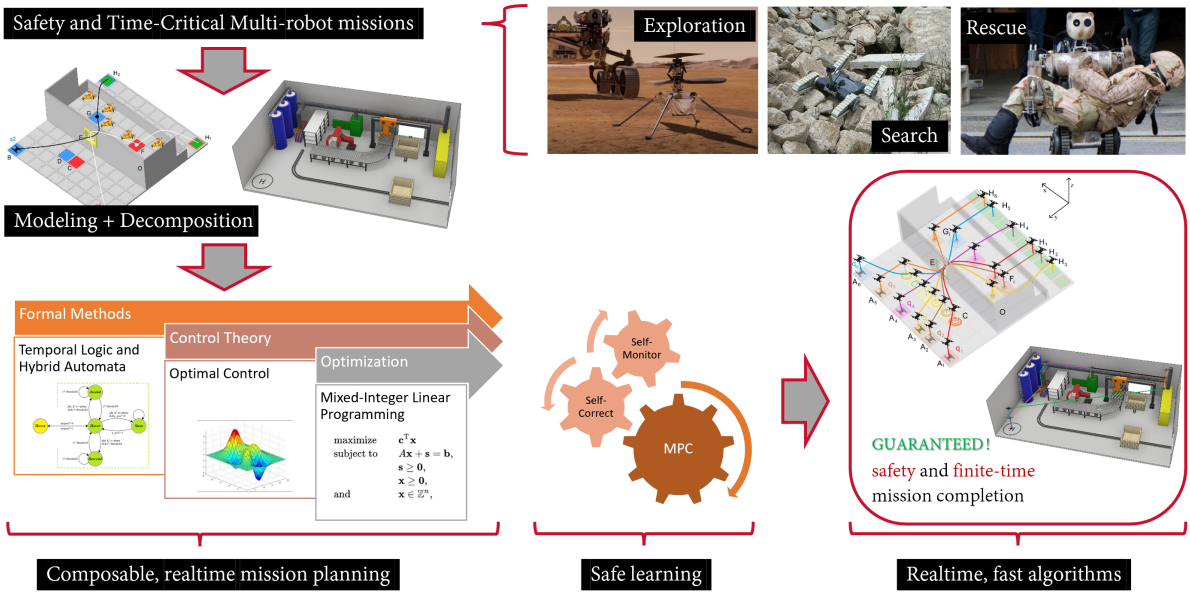


Figure 1.2: Visual overview of the proposed approach.

self-monitoring and self-correction for agents using hybrid automata theory and model predictive control (MPC). Self-monitoring and self-correction refer to the problems in autonomy where the autonomous agents monitor their performance, detect deviations from normal or expected behavior, and learn to adjust both the description of their mission/task and their performance online, to maintain the expected behavior and performance. In this setting, we propose a formal and composable notion of safety and adaptation for autonomous multiagent systems, which we refer to as safe learning. We revisit one of the earlier use cases to demonstrate the capabilities of our approach for a team of autonomous UAVs in a surveillance and search and rescue mission scenario. Figure 1.2 portrays a visual summary of our proposed approach.

In light of the above discussion, the main contributions of this thesis can be listed as follows:

- A composable, safe, hybrid, and realtime mission planning method for multiagent systems with finite-time guarantees. (see Chapter 2)

- A cooperative mission planning method for multiagent systems with distributed consensus dynamics in a leader-follower setting. (see Chapter 3)
- A composable, safe, and realtime mission planning method for UAV-based inspection tasks. (see Chapter 4)
- A safe learning; i.e., self-monitoring and self-correction mechanism for multiagent systems using hybrid automata and event-triggered MPC, which can be used in conjunction with any of the above three methods. (see Chapter 5)

## 1.3 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 details a composable, safe, hybrid, and realtime mission planning method for multiagent systems with finite-time guarantees using metric temporal logic (MTL). We use a UAV-based search and rescue scenario as our case study. In Chapter 3, we describe a cooperative mission planning method for multiagent systems with distributed consensus dynamics in a leader-follower setting. In this chapter, we utilize the example of identifying survivors in case of a natural disaster with a team of ground robots, as our case study. In Chapter 4, we use the example of a pipeline inspection problem in a smart factory environment, to portray a composable, mission planning method for UAV-based inspection tasks. In Chapter 5, we describe a safe learning mechanism for multiagent systems and introduce the ideas of self-monitoring and self-correction. We then revisit one of the earlier use cases to show case the performance of the proposed approach. Finally, in Chapter 6, we conclude with a brief summary of the thesis and a short discussion on future directions and prospects.

# Chapter 2: Composable, Safe, Hybrid, and Realtime Mission Planning for Multi -agent Systems with Finite-Time Guarantees

In this chapter, we propose a hybrid, compositional, optimization-based method for realtime mission planning for quadrotors in a time-critical search and rescue scenario. Starting with a known environment, we specify the mission using metric temporal logic (MTL) and use a hybrid dynamical model to capture the various modes of UAV operation. We then divide the mission into several sub-tasks, by exploiting the invariant nature of safety and timing constraints along the way, and the different modes (i.e., dynamics) of the UAV. For each sub-task, we translate the MTL specifications into linear constraints, and solve the associated optimal control problem for desired path, using a mixed-integer linear program (MILP) solver. The complete path for the mission is constructed recursively, by composing the individual, optimal sub-paths. We show by simulations that the resulting suboptimal trajectories satisfy the mission specifications, and this approach leads to significant reduction in computational complexity of the problem, making it possible to implement in real time.

Our proposed method ensures the safety of UAVs at all times and guarantees finite-time mission completion. It is also shown that our approach scales up nicely for a large number of UAVs under some realistic assumptions on the environment of operation.

Following are the main contributions of this chapter:

- A hybrid optimization-based framework for rescue mission planning for UAVs using MTL specifications under the assumption of known environment, and using a rich, hybrid dynam--ical model for the UAVs.
- Decomposition of complex MTL specifications and their translation into linear constraints.
- Fast (i.e., realtime) and recursive computation of safe, composable, suboptimal trajectories for UAVs with finite-time guarantees.
- Limitations and scalability results for the proposed approach for large number of UAVs in a constrained environment.

## 2.1 Related Work

Given any high level task, it is a standard practice in classic motion planning literature [11], to look for a set of trajectories, which the robot can follow while satisfying the desired task specifications [12]. This gives rise to the notion of optimal path planning [13], which considers an optimal path in the sense of optimizing some suitable cost function and finding a control law, to go from one position to another while satisfying some constraints [14]. Traditionally, methods such as potential functions [15, 16], have been used for multiagent mission planning. Although these methods are easy to compute, they often suffer from problems like the agents getting stuck in local-minima, or their inability to find a feasible path in areas where the density of obstacles is relatively high [17].

Many trajectory planning methods in literature focus on obstacle and collision avoidance (between the agents) as their primary objective [18, 19], which is well-suited for applications

like crowd simulation and robotic swarms [20, 21]. These methods are shown to be efficient in tackling dynamic environments as well as a large number of agents [22], and they are able to compute safe trajectories for several agents in a reasonably fast time [23]. However, they are not suitable for situations where a multiagent mission imposes some finite-time constraints on the team of robots, and requires them to execute some specific tasks in parallel, in addition to going from one point to the other while avoiding obstacles and their fellow agents. Cooperative aerial surveying [24], and multiagent search and rescue mission planning [25] are two examples of such tasks, where some sort of guarantees are desired on the time that is required to complete the mission, in addition to the safety of the agents themselves.

Temporal logic (TL) [26] seems to address this problem, since it enables us to specify complex tasks in compact mathematical form. A bulk of modern motion planning literature is based on linear temporal logic (LTL) [27], which is useful for specifying tasks such as visiting certain objectives periodically and surveying areas [28], and and ensuring stability and safety etc. [29, 30]. These LTL-based mission planners are usually easier to compute and faster to implement. However, from a control theory perspective, LTL only accounts for timing in the infinite-horizon sense i.e., it can only guarantee something will *eventually* happen over an infinite, discrete sequence of time. Finite LTL or LTLf [31] is a variant of LTL that can be interpreted over finite, discrete-time sequences, however it is not rich enough to describe finite, realtime constraints.

On the other hand, metric temporal logic (MTL) [32, 33], can express finite-time require--ments between various events of the mission as well as on each event duration. This allows us to specify safety-critical missions with dynamic task specifications and finite-time constraints. An optimization-based method for LTL tasks was proposed in [34], and was extended in [35], where

the authors translate LTL specifications to mixed-integer linear programming (MILP) constraints, which are then used to solve an optimal control problem for a linear point-robot model. This work was further extended in [36], where the authors used bounded-time temporal constraints using extended-LTL, for motion planning with linear system models. However, all these works did not incorporate a rich dynamical model of the robot, and also illustrated significant computational complexity issues for the proposed methods in case of planning for multiple robots in 3D space.

Recently, optimization-based methods with MTL specifications for single [37] and multiple [38] robots, do guarantee safe and finite-time mission completion. However, in both cases, the computation of the optimal trajectory is very expensive (in the order of $\sim$500s computation time or more), and hence cannot be implemented in real time. Moreover, these works put high constraints on the robot maneuverability, by limiting its dynamics to a simple linear (point-robot) model, which contradicts with the primary reason for deploying quadrotors in constrained dynamic environments. More recently, the use of signal temporal logic (STL) for event-triggered [39], and risk-aware [40] planning and control using very simple robot models has also been presented. While STL provides a good measure of robustness in space and time for planning complex tasks, its computational complexity is a lot worse in general than MTL, using any of the standard planning methods. Therefore, in this thesis, unless specified otherwise, we will stick with only MTL for planning complex missions and designing control policies for multiagent systems, because it is a good compromise between LTL and STL, in terms of its capabilities and performance.

In this chapter, therefore, our intention is to use the rich dynamics of the robots in an intelligent way to divide and conquer the computationally complex problem of mission planning for multiple UAVs using finite-time MTL constraints.

## 2.2 Notation and Preliminaries

In this section, we describe some mathematical notation and preliminaries that are followed throughout, in the rest of this chapter and much of this thesis.

### 2.2.1 System Dynamics

Any dynamical system can be represented in the form:

$$\dot{x}(t) = f(t, x, u)$$

where for all time $t$ continuous

- $x(t) \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state vector of the system
- $x_0 \triangleq x(0) \in \mathcal{X}_0 \subseteq \mathcal{X}$ is the initial condition of the state vector and
- $u(t) \in \mathcal{U} \subset \mathbb{R}^m$ is the set of control inputs which is constrained in the control set $\mathcal{U}$.

Given a nonlinear model of the system, a linearization around an operating point or trajectory $(x_*(t), u_*(t))$ is expressed as:

$$\dot{\hat{x}}(t) = A(t)\hat{x}(t) + B(t)\hat{u}(t)$$

where for all time $t$ continuous

- $\hat{x}(t) = x(t) - x_*(t)$
- $\hat{u}(t) = u(t) - u_*(t)$

- $A(t) = \frac{\partial f}{\partial x}(t)\Big|_{x=x_*, u=u_*}$
- $B(t) = \frac{\partial f}{\partial u}(t)\Big|_{x=x_*, u=u_*}$

Any discrete time (possibly nonlinear) dynamical system can be represented in the form:

$$x(t+1) = f(t, x(t), u(t)) \tag{2.1}$$

where $x(t) \in \mathcal{X}$ is the state vector, $x(0) \in \mathcal{X}_0 \subseteq \mathcal{X}$ is the initial condition of the state, and $u(t) \in \mathcal{U} \subset \mathbb{R}^m$ is the set of control inputs for all $t = 0, 1, 2, \cdots$. Let us denote the trajectory for System (2.1), with initial condition $x_0$ at $t_0$, and input $u(t)$ as: $\mathbf{x}_{t_0,x_0,u(t)} = \{x(t) \mid t \geq t_0, x(t+1) = f(t, x(t), u(t)),\ x(t_0) = x_0\}$. For the sake of convenience, we use the shorthand notation i.e., $\mathbf{x}_{t_0}$ instead of $\mathbf{x}_{t_0,x_0,u(t)}$ to represent system trajectory whenever no explicit information about $u(t)$ and $x_0$ is required. As is in the usual control literature setting, a linearization of the discrete time dynamics for System (2.1) about a given trajectory (or operating point) looks like:

$$\hat{x}(t+1) = A(t)\hat{x}(t) + B(t)\hat{u}(t) \tag{2.2}$$

for all $t = 0, 1, 2, \cdots$. We use System (2.2) form dynamics in our problem formulation.

### 2.2.2 Metric Temporal Logic (MTL)

The convention on MTL syntax and semantics followed in this chapter/thesis is the same as presented in [32].

***Definition* 1.** *(Atomic proposition) An atomic proposition is a statement with regard to the state variables $x$ that is either* **True** $(\top)$ *or* **False** $(\bot)$ *for some given values of $x$.[41]*

Let $\Pi = \{\pi_1, \pi_2, \cdots \pi_n\}$ be the set of atomic propositions which labels $\mathcal{X}$ as a collection of the areas of interest in some workspace, which can possibly be time varying. Then, we can define a map $L$ which labels this workspace or environment as follows:

$$L : \mathcal{X} \times \mathcal{I} \to 2^{\Pi}$$

where $\mathcal{I} = \{[t_1, t_2] \mid t_2 > t_1 \geq 0\}$ and $2^{\Pi}$ denotes the power set of $\Pi$ as usual. In general, $\mathcal{I}$ represents an interval of time but it may just also represent a time instance. For each trajectory of System (2.2) i.e., $\mathbf{x}_{t_0}$ as before, the corresponding sequence of atomic propositions, which $\mathbf{x}_{t_0}$ satisfies is given as: $\mathcal{L}(\mathbf{x}_0) = L(x(0), 0)L(x(1), 1)....$

We later specify the tasks formally using MTL formulae, which can incorporate finite timing constraints. These formulae are built on the stated atomic propositions (Definition 1) by following some grammar.

***Definition* 2.** *(MTL syntax) The syntax of MTL formulas are defined in accordance with the following rules of grammar:*

$$\phi ::= \top \mid \pi \mid \neg\phi \mid \phi \vee \phi \mid \phi \mathbf{U}_I \phi$$

where $I \subseteq [0, \infty]$, $\pi \in \Pi$, $\top$ and $\neg\top (= \bot)$ are the Boolean constants for $true$ and $false$ respectively. $\vee$ represents the disjunction while $\neg$ represents the negation operator. $\mathbf{U}_I$ denotes the Until operator over the time interval $I$. Similarly, other operators (both boolean and temporal) can be expressed using the grammar imposed in Definition 2. Some examples are conjunction ($\wedge$), always on $I$ ($\Box_I$), eventually within $I$ ($\Diamond_I$) etc. Further examples of temporal operators can be found in [34].

***Definition* 3.** *(MTL semantics) The semantics of any MTL formula $\phi$ is recursively defined over a trajectory $x_t$ as:*

$x_t \models \pi$ *iff* $\pi \in L(x(t), t)$

$x_t \models \neg\pi$ *iff* $\pi \notin L(x(t), t)$

$x_t \models \phi_1 \vee \phi_2$ *iff* $x_t \models \phi_1$ *or* $x_t \models \phi_2$

$x_t \models \phi_1 \wedge \phi_2$ *iff* $x_t \models \phi_1$ *and* $x_t \models \phi_2$

$x_t \models \bigcirc\phi$ *iff* $x_{t+1} \models \phi$

$x_t \models \phi_1 \mathbf{U}_I \phi_2$ *iff* $\exists t' \in I$ *s.t.* $x_{t+t'} \models \phi_2$ *and* $\forall\, t'' \leq t'$,

$x_{t+t''} \models \phi_1$.

Thus, for instance, the expression $\phi_1 \mathbf{U}_I \phi_2$ means the following: $\phi_2$ is true within time interval $I$, and until $\phi_2$ is true, $\phi_1$ must be true. Similarly, the MTL operator $\bigcirc\phi$ means that $\phi$ is true at next time instance. $\Box_I \phi$ means that $\phi$ is always true for the time duration or during the interval $I$, $\Diamond_I \phi$ implies that $\phi$ eventually becomes true within the interval $I$. More details on specifying tasks as MTL formulae can also be found in [33]. Similar to the convention used in Definition 3, a system trajectory $\mathbf{x}_{t_0}$ satisfying an MTL specification $\phi$ is denoted as $\mathbf{x}_{t_0} \models \phi$. This is the general temporal constraint representation, which we use later in our optimal control formulation.

More complicated formulas can be specified using a variety of compositions of two or more MTL operators. For example, $\Diamond_{I_1} \Box_{I_2} \phi$ suffices to the following: within time interval $I_1$, $\phi$ will be eventually true and from that time instance, it will always be true for an interval or duration of $I_2$. The remaining Boolean operators such as implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$) can also be represented using the grammar rules and semantics given in Definition 2 and Definition 3.

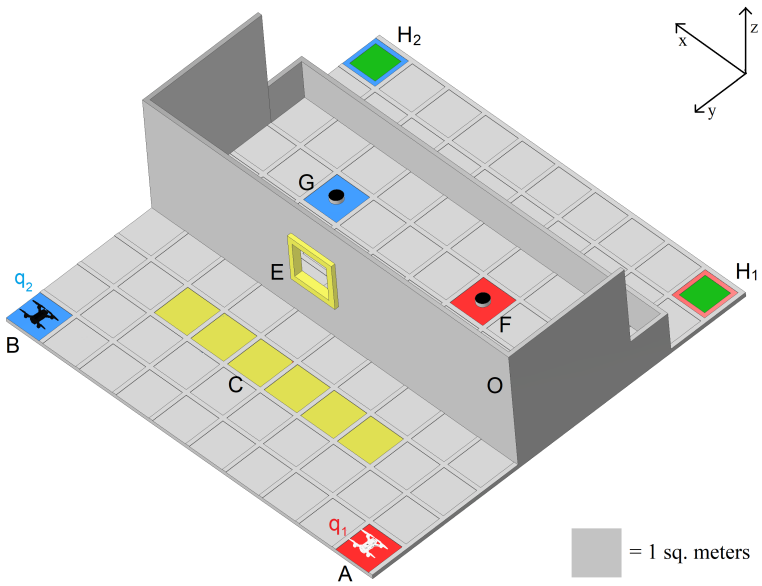


Figure 2.1: CAD model for the workspace used. The environment is a 10x10x3 $m^3$ workspace which is divided into several 2D regions of interest that are labeled with alphabets and marked with different colors.

### 2.2.3 The Workspace

Throughout this chapter, we repeatedly refer to a time-critical rescue mission defined on the constrained workspace shown in Fig. 2.1. It is a custom built CAD environment, designed with the intention to closely suit our problem. As shown, various areas of interest are marked on the workspace using different alphabets. The role of each of these areas will become obvious as we formulate the problem and specify the mission using MTL specifications later in this chapter. For now, we briefly describe the mission in a two UAV setting.

Starting from initial positions $A$ and $B$, two quadrotor UAVs $q_1$ and $q_2$ need to rescue two objects located at $F$ and $G$ respectively from a constrained environment. The objects are accessible only through a window $E$, with dimensions such that it allows only one UAV to pass at a given time. Therefore, one of the UAVs has to wait at $C$ for the other to pass first. It is assumed that there are no additional obstacles in the area other than the walls $O$ and the UAVs

themselves. The task for each UAV is to grasp its respective target object, and transport it to safety (marked $H_1$ and $H_2$ respectively for $q_1$ and $q_2$) in given finite time. In doing so, the UAVs need to avoid the obstacles $O$ as well as each other, in particular at the window $E$. Later on, we also use prime region notation; e.g., $A'$ to represent the same 2D region $A$. There, $A'$ represents an altitude (w.r.t. $z$-axis) variation of the quadrotor while it is in the same 2D region $A$.

## 2.3 Quadrotor Dynamics

We adopt the generalized nonlinear model for the quadrotor presented in [42]. We build a hybrid model for the system with five linear modes, namely *Take off*, *Land*, *Hover*, *Steer*, and a task specific *Grasp* mode. The linearization for each mode is carried out separately about a different operating point. This enables our system to have rich dynamics with less maneuverability restrictions, while each mode still being linear. This is an important point and its significance becomes apparent once we formulate the problem, since it requires all constraints to be linear.

### 2.3.1 General Nonlinear Model

The dynamics of a quadrotor can be fully specified using two coordinate frames. One is a fixed earth (or world) frame, and the second is a moving body frame. Let the homogeneous transformation matrix from body frame to earth frame be $R(t)$, which is a function of time $t$. In state space representation, the quadrotor dynamics are represented as twelve states namely $[x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, \omega_\phi, \omega_\theta, \omega_\psi]^T$, where $\xi = [x, y, z]^T$ and $v = [v_x, v_y, v_z]^T$ represent the position and velocity of the quadrotor respectively with respect to the body frame. $[\phi, \theta, \psi]^T$

are the angles along the three axes (i.e., roll, pitch, and yaw respectively), and $\Omega = [\omega_\phi, \omega_\theta, \omega_\psi]^T$ represents the vector containing their respective angular velocities. Under the rigid body assump--tions on its airframe, the Newton-Euler formalism for quadrotor in earth frame is given by:

$$\begin{aligned} \dot{\xi} &= v \\ \dot{v} &= -g\mathbf{e_3} + \frac{F}{m} R\mathbf{e_3} \\ \dot{R} &= R\hat{\Omega} \\ \dot{\Omega} &= J^{-1}(-\Omega \times J\Omega + \tau) \end{aligned} \tag{2.3}$$

where $J$ is the moment of inertia matrix for the quadrotor, $g$ is the gravitational acceleration, $\mathbf{e_3} = [0, 0, 1]^T$, $F$ is the total thrust produced by the four rotors, and $\tau = [\tau_x, \tau_y, \tau_z]^T$ is the torque vector, whose components are the torques applied about the three axes. So, $F$, $\tau_x$, $\tau_y$, and $\tau_z$ are the four control inputs to System (2.3).

### 2.3.2 Hybrid Model with Linear Modes

System (2.3) serves as a starting point for generating a hybrid model for the quadrotor with five modes, which are represented by a labeled transition system as shown in Fig. 2.2. As usual, the states (or modes) denote the action of the UAV, such as *Take off* and *Steer*, while the edges represent the change or switch to another action. The change is governed by some suitable guard condition. Note, that some edges do not exist; for example, the quadrotor cannot go from *Land* to *Hover* without taking the action *Take off*. Each state of the transition system follows certain dynamics, which result from a linearization of System (2.3) around a different operating point. For example, consider the *Hover* mode. One possible choice of operating points for the

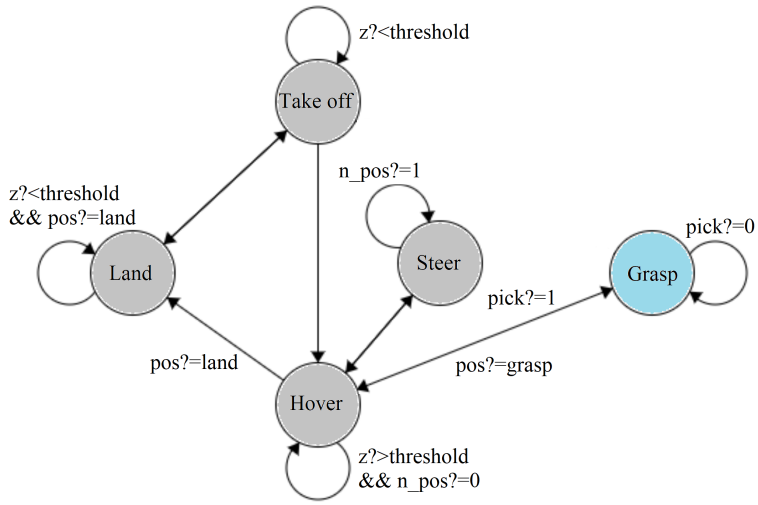


Figure 2.2: The simplified hybrid dynamical model for the quadrotor. Some guard conditions are hidden for readability. We use linearized dynamics around different operating points for each mode. This makes the model rich in dynamics as well as linear at the same time.

linearization of system dynamics in this mode is $\psi = 0$. This implies that the two states $\psi$ and $\omega_\psi$ i.e., the yaw angle and its respective angular velocity are identically zero, and thus can be removed from the state space representation. Consequently, the state space dimension is reduced to ten, and the control set is reduced to three inputs as well; i.e., $F, \tau_x$, and $\tau_y$. The resulting linearized model can be written in standard (discrete time) form as: $\sigma(t+1) = A(t)\sigma(t) + B(t)\gamma(t)$, where $\sigma(t)$ is the state, and $\gamma(t)$ is the input (in vector notation), with the two system matrices given as:

$$A = \begin{bmatrix} \mathbf{0} & I & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \begin{bmatrix} 0 & g \\ -g & 0 \\ 0 & 0 \end{bmatrix} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & I \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}; \quad B = \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \begin{bmatrix} 0 \\ 0 \\ 1/m \end{bmatrix} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & I_{2\times 3} J^{-1} \end{bmatrix}$$

where $I_{2,3} = [I_{2,2} \quad \mathbf{0}_{2,1}]$, and all zero and identity matrices in $A(t)$ and $B(t)$ are of proper

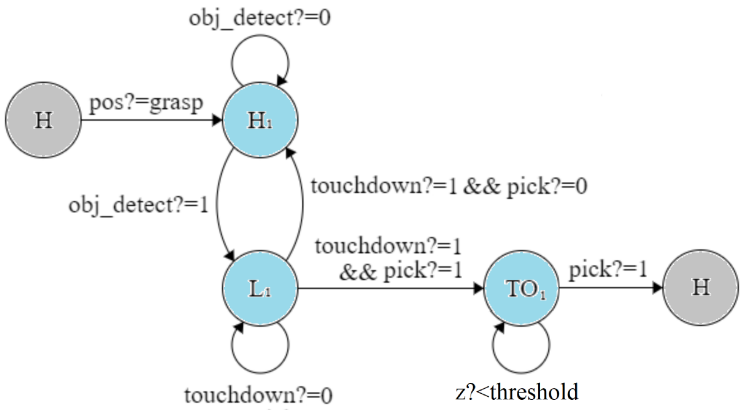


Figure 2.3: The *Grasp* mode expressed as a combination of *Hover* ($H_1$), *Land* ($L_1$), and *Take off* ($TO_1$) modes (colored cyan), with special guard conditions.

dimensions. We adopt similar procedure to linearize System (2.3) around other operating points for different modes, and obtain linearized dynamics for the hybrid model. More details about the selection of these operating points for different modes can be found in [43] and [44].

### 2.3.3 The Grasp Mode

Grasping in general is a very challenging problem, in particular when dexterity-based manipulation is involved. Since *Grasp* is the only task specific dynamical mode of our system, it was advisable to simplify the grasping routine within the high level task. However, in case of aerial grasping, some passive mechanisms [45, 46] have been shown to be very reliable in grasping an object with an instantaneous touchdown onto its surface [47]. Under this reliable passive aerial grasping assumption (i.e., instantaneous touchdown and grasp), we can express the *Grasp* mode as a switching combination of *Hover*, *Land*, and *Take off* dynamics with special guard conditions. This clearly simplifies the problem of having the need to introduce a complex gripper and its end-effector dynamics into the *Grasp* mode of the hybrid model. Figure 2.3 depicts this representation of the *Grasp* mode in terms of the *Hover*, *Land*, and *Take off* dynamics.

## 2.4 Method: Formulation and Solution

Given the map of the environment, we can write down a mission specification for each quadrotor as an MTL formula $\phi_i$. For the workspace described earlier in this chapter, a possible MTL mission specification for the $i^{th}$ quadrotor $q_i$ can be written as:

$$\phi_i = \Diamond_{[0,T_1]}(Object\ Location) \wedge \Box_{[0,T_2]}(Object\ Location)'$$

$$\wedge \Diamond_{[0,T_3]} \Box(Safe\ Location) \wedge \Box\neg(Obstacles) \wedge \Box\neg q_j$$

where $i, j \in \{1, 2, ..., N\}$, $i \neq j$, and $q_j \in \mathcal{N}_i(t)$. $\mathcal{N}_i(t)$ represents the neighborhood set of quadrotor $q_i$ i.e., all quadrotors $q_j$ s.t. $||q_i - q_j|| \leq \rho$, for some $\rho > 0$. $T_3$, $T_2$, $T_1$ are discrete time units, and the prime notation models the grasping action within the same area of the workspace with some altitude variation of the UAV along the $z$-axis. Thus, for instance, in the two UAV setting as shown in Fig. 2.1, a possible MTL specification for quadrotor $q_1$ is:

$$\phi_1 = \Diamond_{[0,T_1]}(F) \wedge \Box_{[0,T_2]}(F)' \wedge \Diamond_{[0,T_3]} \Box(H_1) \wedge \Box\neg(O) \wedge \Box\neg q_2$$

A similar mission specification $\phi_2$ can be written for the quadrotor $q_2$ and so on. Now using the quadrotor dynamics and these MTL formulae $\phi_i$, we can state the rescue mission planning as an optimal control problem.

### 2.4.1 Problem Statement and Formulation

We set up the described mission as a standard optimal control problem in discrete time. Given the system dynamics (2.2), the objective is to find a suitable control law that steers each

quadrotor through some regions of interest in the workspace within desired time bounds, so that it evacuates the target safely to a desired location. This control also optimizes some cost function, while the associated task constraints are specified by an MTL expression.

As before, let $\phi_i$ denote the MTL specification, and $J(x_i(t, u_i(t)), u_i(t))$ be the cost function to be minimized. Then, the corresponding optimization problem for quadrotor $q_i$, $i \in \{1, 2, ..., N\}$ is given by:

**Problem 1.**

$$\begin{aligned} \min_{x_i, u_i} \quad & J(x_i(t, u_i(t)), u_i(t)) \\ \textit{s.t.} \quad & x_i(t+1) = A(t)x_i(t) + B(t)u_i(t) \\ & \mathbf{x_i}_{t_0} \models \phi_i \end{aligned}$$

Problem 1 is a discrete time optimal control problem with linear dynamics. However, it includes a complex MTL satisfiability constraint. In addition, notice that in our case, the hybrid model of the quadrotor has multiple linear modes. Therefore, in its current form, Problem 1 is not directly solvable. So, intuitively it would make sense to break down the problem into sub-problems for each dynamical mode of the system.

We now describe two methods that transform Problem 1 into a set of readily solvable mixed-integer linear programs (MILPs). First, we describe a method for translating the MTL satisfiability constaint into a set of linear constraints. This approach renders Problem 1 solvable as a MILP for a linear cost and a given dynamical mode of the system. Secondly, we decompose the complex MTL specifications $\phi_i$ into a set of simpler MTL formulae $\phi_i^k$, $i \in \{1, 2, ..., N\}$ and $k \in \{1, 2, ..., M\}$. This suffices to breaking down the original problem into $M$ sub-problems, each with a set of linear constraints and exactly one associated dynamical mode of the hybrid system. Combining both these methods, the resulting $M$-MILPs are expected to have significantly reduced

computational complexity than the parent problem.

### 2.4.2 MTL Formulae to Linear Constraints

This method is based on the approach presented in [34] where the authors translate LTL specifications into linear constraints. Along similar lines, we now present an approach to translate MTL specifications into mixed integer linear constraints using inspiration from the steps presented in [37]. We start with a simple temporal specification and work through the procedure to convert it to a set of mixed integer linear constraints. We then use this example as a foundation for translating the MTL operators into equivalent linear constraints.

Consider the constraint that a trajectory $x(t)$ lies within a convex polytope $\mathcal{K}$ at time $t$. Since $\mathcal{K}$ is convex, it can be represented as an intersection of a finite number of halfspaces. A halfspace can be represented as set of points, $\mathcal{H}_i = \{x : h_i^T x \leq a_i\}$. Thus, $x(t) \in \mathcal{K}$ is equivalent to $x(t) \in \cap_{i=1}^n \mathcal{H}_i = \cap_{i=1}^n \{x : h_i^T x \leq a_i\}$. So, the constraint $x(t) \in \mathcal{K} \, \forall \, t \in \{t_1, t_1+1, \cdots t_1+n\}$ can be represented by the set of linear constraints $\{h_i^T x(t) \leq a_i\}$, $\forall \; i = \{1, 2, \cdots, n\}$ and $\forall t \in \{t_1, t_1+1, \cdots t_1+n\}$.

In a polytopic environment, atomic propositions (see Definition 2), $p, q \in \Pi$, are related to state of the system via conjunction and disjunction of linear halfspaces [34]. Let us consider the case of a convex polytope and let $b_i^t \in \{0, 1\}$ be some binary variables associated with the corresponding halfspaces $\{x(t) : h_i^T x(t) \leq a_i\}$ at time $t = 0, ..., N$. We can then force the constraint: $b_i^t = 1 \iff h_i^T x(t) \leq a_i$, by introducing the following linear constraints:

$$h_i^T x(t) \leq a_i + M(1 - b_i^t) \tag{2.4}$$

$$h_i^T x(t) \geq a_i - M b_i^t + \epsilon$$

where $M$ and $\epsilon$ are some large and small positive numbers respectively. If we denote $K_t^{\mathcal{K}} = \wedge_{i=1}^n b_i^t$, then $K_t^{\mathcal{K}} = 1 \iff x(t) \in \mathcal{K}$. This approach is extended to the general nonconvex case by convex decomposition of the polytope. Then, the decomposed convex polytopes are related using disjunction operators. Similar to conjunction, as is described later in this section, the disjunction operator can also be translated to mixed integer linear constraints.

Let $S_\phi(x, b, u, t)$ denote the set of all mixed integer linear constraints corresponding to a temporal expression $\phi$. Using the described procedure, once we have obtained $S_p(x, b, u, t)$ for atomic propositions $p \in \Pi$, we can formulate $S_\phi(x, b, u, t)$ for any MTL formula $\phi$. Now, for the Boolean MTL operators, such as $\neg$, $\wedge$, $\vee$, let $t \in \{0, 1, ..., N\}$, and as before, let $K_t^\phi \in [0, 1]$ be the continuous variables associated with the formula $\phi$ generated at time $t$ with atomic propositions $p \in \Pi$. Then $\phi = \neg p$ is the negation of an atomic proposition, and it can be modeled as:

$$K_t^\phi = 1 - K_t^p \tag{2.5}$$

the conjunction operator, $\phi = \wedge_{i=1}^m p_i$, is modeled as:

$$\begin{aligned} K_t^\phi &\leq K_t^{p_i}, \quad i = 1, ..., m \\ K_t^\phi &\geq 1 - m + \sum_{i=1}^m K_t^{p_i} \end{aligned} \tag{2.6}$$

and the disjunction operator, $\phi = \vee_{i=1}^{m} p_i$, is modeled as:

$$K_t^{\phi} \geq K_t^{p_i}, \quad i = 1, ..., m \tag{2.7}$$
$$K_t^{\phi} \leq \sum_{i=1}^{m} K_t^{p_i}$$

Similar to binary operators, temporal operators such as $\diamond$, $\square$, and $\mathcal{U}$ can be modeled using linear constraints as well. Let $t \in \{0, 1, ..., N - t_2\}$, where $[t_1, t_2]$ is the time interval used in the MTL specification $\phi$. Then, eventually operator: $\phi = \diamond_{[t_1,t_2]} p$ is modeled as:

$$K_t^{\phi} \geq K_{\tau}^{p}, \quad \tau \in \{t + t_1, ..., t + t_2\} \tag{2.8}$$
$$K_t^{\phi} \leq \sum_{\tau=t+t_1}^{t+t_2} K_{\tau}^{p}$$

and always operator: $\phi = \square_{[t_1,t_2]} p$ is represented as:

$$K_t^{\phi} \leq K_{\tau}^{p}, \quad \tau \in \{t + t_1, ..., t + t_2\} \tag{2.9}$$
$$K_t^{\phi} \geq \sum_{\tau=t+t_1}^{t+t_2} K_{\tau}^{p} - (t_2 - t_1)$$

and so on (for more details see [34]).

and until operator: $\phi = p\,\mathcal{U}_{[t_1,t_2]}\,q$ is equivalent to:

$$
\begin{aligned}
& c_{tj} \leq K_q^j \quad j \in \{t+t_1, \cdots, t+t_2\} \\
& c_{tj} \leq K_p^l \quad l \in \{t, \cdots, j-1\}, j \in \{t+t_1, \cdots, t+t_2\} \\
& c_{tj} \geq K_q^j + \sum_{l=t}^{j-1} K_p^l - (j-t) \quad j \in \{t+t_1, \cdots, t+t_2\} \\
& c_{tt} = K_q^t \\
& K_t^\phi \leq \sum_{j=t+t_1}^{t+t_2} c_{tj} \\
& K_t^\phi \geq c_{tj} \quad j \in \{t+t_1, \cdots, t+t_2\}
\end{aligned}
\tag{2.10}
$$

The equivalent linear constraints for until operator (2.10) are constructed using a procedure similar to [34]. The modification for MTL comes from the following result in [32].

$$
K_t^\phi = \bigvee_{j=t+t_1}^{t+t_2} \left( (\wedge_{l=t}^{l=j-1} K_p^l) \wedge K_q^j \right).
$$

All other combinations of MTL operators for example, eventually-always operator: $\phi = \Diamond_{[t_1,t_2]}\Box_{[t_3,t_4]}p$ and always-eventually operator: $\phi = \Box_{[t_1,t_2]}\Diamond_{[t_3,t_4]}p$ etc., can be translated to similar linear constraints using (2.5)-(2.10). In addition to the collective operator constraints, we need another constraint $K_0^\phi = 1$ as well, which suffices to the overall satisfaction of a task specification $\phi$.

Using this approach, we can translate an MTL formula $\phi$ into a set of mixed integer linear constraints $S_\phi(x, b, u, t)$, which converts the associated optimal control problem (e.g. Problem 1) to a MILP for some linear cost function.

### 2.4.3 Decomposition of Complex MTL Formulae

Notice that the worst case complexity of the above MILP (Problem 1 with linear constraints and linear cost) is exponential i.e., $O(2^{mT})$, where $m$ is the number of boolean variables or equivalently the number of halfspaces required to express the MTL formula, and $T$ is the discrete time horizon. Therefore, it is logical to consider decomposing the task specification $\phi_i$ into several simpler sub-tasks $\phi_i^k$, where $i \in \{1, 2, ..., N\}$ and $k \in \{1, 2, ..., M\}$.

In [48], the authors proposed a timed-automata-based approach to decompose a complex LTL specification into a finite number of simpler LTL specifications. In case of MTL, this method is not applicable in general, because MTL to Buchi-automata conversion is not always possible. However, in the special case, where the finite-time constraints are specified as intervals i.e., in case of metric interval temporal logic (MITL), we can break down a complex MTL specification (mission) into finite number of simpler MTL specifications (sub-tasks), if the sum of the finite timing intervals in decomposed MTL specifications does not violate the finite timing interval of the original MTL specification. We can state this proposition as the following theorem:

***Theorem* 1.** *Given an MITL specification $\phi_i$, there exists some finite $M$-length decomposition $\phi_i^k$, $k \in \{1, 2, ..., M\}$, s.t. $\wedge_{k=1}^{M}(\phi_i^k) \implies \phi_i$, if $\sum_{k=1}^{M} T_k \leq T_i$, where $T_i$ is the finite timing interval for $\phi_i$, and $T_k$s are the corresponding finite timing intervals for $\phi_i^k$, $\forall k \in \{1, 2, ..., M\}$.*

Theorem 1 is based on the work of Schillinger [49], where the authors present a decompos--ition method for LTLf specifications. The proof for discrete-time MITL specifications follows directly from there. For the case of continuous i.e., realtime MITL specifications, it is readily verified using a timed-automata simulation in UPAAL [50]. For more details on the proof, refer to [49] and [44].

Notice that Theorem 1 does not guarantee an equivalence relationship between $\phi_i$ and $\phi_i^k$. Moreover, this decomposition is not unique. However, if the mission is specified by an MITL specification and the system is represented by a hybrid model, which is the case in our problem, one convenient design choice is to pick such $\phi_i^k$, for which there is only one associated dynamical mode of the system. For example, consider again the workspace in Fig. 2.1 and let $\phi_1$ be given by:

$$\phi_1 = \Diamond_{[0,20]}(F) \wedge \Box\neg(O)$$

then, a possible decomposition $\phi_1^k$ is given by:

$$\phi_1^1 = \Box(A) \wedge \Diamond_{[0,5]}(A)' \quad [mode : \textit{Take off}]$$

$$\phi_1^2 = \Diamond_{[0,5]}(C) \wedge \Box\neg(O) \quad [mode : \textit{Steer}]$$

$$\phi_1^3 = \Diamond_{[0,9]}(F) \wedge \Box\neg(O) \quad [mode : \textit{Steer}]$$

which clearly satisfies Theorem 1.

In addition, a closer inspection of $\phi_i$ in our case reveals that all safety and timing constraints specified by $\phi_i$ need not be satisfied by the UAV during the whole mission. There are some critical safety constraints such as "always avoid $O$" that need to be satisfied most of the times, but majority of the constraints are purely local, based on the position of the robot in the workspace. For example, in the two UAV setting, a timing constraint for quadrotor $q1$ to grasp the object at location $F$ within 10 time units is independent from (or invariant of) a timing constraint for it to reach location $C$ from $A$ within 5 time units, and so on.

### 2.4.4 Final Trajectory Generation

By decomposing the mission specification $\phi_i$, and by using the MTL to linear constraints translation mechanism, we can replace Problem 1 with a collection of smaller optimization problems, each with a sub-task specification represented as an MTL formula $\phi_i^k$, and an associated linear mode of the hybrid model.

Here, the linear cost function of choice is $\sum_{t=0}^{T} |u_i(t)|$, where $T$ is the discrete time horizon for the optimal trajectory. Thus, our final formulation of the problem is given by:

**Problem 2.**

$$\begin{aligned} \min_{x_i, u_i} \quad & \sum_{t=0}^{T} |u_i(t)| \\ \textit{s.t.} \quad & x_i(t+1) = A_l(t)x_i(t) + B_l(t)u_i(t) \\ & \mathbf{x_i}_{t_0} \models \phi_i^k \end{aligned}$$

where $\phi_i^k$ is the MTL specification for the $k^{th}$ sub-task for the $i^{th}$ UAV, $A_l(t)$, $B_l(t)$ are the linear system matrices for the $l^{th}$ mode, and $\mathbf{x_i}_{t_0}$ is the resulting optimal trajectory for the $k^{th}$ sub-task, with $i \in \{1, 2, 3, ..., N\}$, $k \in \{1, 2, 3, ..., M\}$, and $l \in \{1, 2, 3, ..., 5\}$.

For example, in the two UAV setting, for quadrotor $q1$, one sub-task is to go from $A$ to $C$ in 5 time units. The MTL specification for this sub-task is given by $\phi_{sub_1^k} = \diamondsuit_{[0,5]}(C) \wedge \square\neg(O)$, and the associated dynamics are selected from the *Steer* mode.

Problem 2 represents a collection of MILPs, which can be solved recursively and efficiently using a MILP solver. The resultant trajectories are locally optimal for each individual sub-task, and their existence inherently guarantees safety and finite-time completion of the respective sub-tasks. The final trajectory for the complete mission is generated over time, by composing all the individual optimal sub-task trajectories. The final path is therefore not optimal but suboptimal

with respect to the original mission specification $\phi_i$ for the $i^{th}$ UAV. However, despite this loss of global optimality, the advantages achieved in terms of reduction in computational complexity, and improved scalability with respect to the number of UAVs are far more important, as is shown by simulation results.

## 2.5 Simulations and Results

We apply the proposed method for solving Problem 2 in the same workspace as shown in Fig. 2.1. The experiments are run through YALMIP-CPLEX solver using MATLAB interface on an Intel NuC. It is portable computer with an Intel core i7 @ 3.7 GHz CPU, an integrated Intel Iris GPU, and 16 GBs of memory. This setup is directly transferable to a quadrotor as a companion module for onboard computation.

We use a $2m$ neighborhood set threshold for the UAVs i.e., $\rho = 2m$. The discrete time horizon for simulation is $T = 30$, and the UAV altitude limit in *Hover* and *Steer* modes is set to $1.5m$. All dynamics are uniformly discretized at a rate of 5 Hz.

### 2.5.1 Case Study I: Validation (2 UAVs)

In the two UAV setting, for the mission $\phi_1$, the sub-tasks for the quadrotor $q_1$ are specified as follows:

$$\phi_1^1 = \Box(A) \wedge \Diamond_{[0,5]}(A)' \quad [mode : \textit{Take off}]$$

$$\phi_1^2 = \Diamond_{[0,5]}(C) \wedge \Box\neg(O) \wedge \Box\neg(q_2) \quad [mode : \textit{Steer}]$$

$$\phi_1^3 = \Diamond_{[0,10]}(F) \wedge \Box\neg(O) \wedge \Box\neg(q_2) \quad [mode : \textit{Steer}]$$

$$\phi_1^4 = \Box(F) \wedge \Diamond_{[0,10]}(F)' \quad [mode : \textit{Grasp}]$$

$$\phi_1^5 = \Diamond_{[0,10]}(H_1) \wedge \Box\neg(O) \wedge \Box\neg(q_2) \quad [mode : \textit{Steer}]$$

$$\phi_1^6 = \Box(H_1) \quad [mode : \textit{Land}]$$

Using the convention defined earlier, the specification $\phi_1^1$ requires the quadrotor $q_1$ to attain desired threshold altitude (represented as $A'$) while staying inside the 2D region marked $A$. $\phi_1^2$ requires the quadrotor $q_1$ to reach $C$ within 5 time units, and $\phi_1^3$ requires it to reach $F$ within 10 time units, while avoiding the obstacle $O$ and the neighboring quadrotor $q_2$. $\phi_1^4$ requires the UAV to grasp the object at $F$ within 10 time units while staying in $F$, whereas $\phi_1^5$ asks it to reach $H_1$ within 10 time units. Finally $\phi_1^6$ forces $q_1$ to stay at $H_1$ indefinitely. The sub-tasks for quadrotor $q_2$ are specified in a similar fashion.

$$\phi_2^1 = \Box(B) \wedge \Diamond_{[0,5]}(B)' \quad [mode : \textit{Take off}]$$

$$\phi_2^2 = \Diamond_{[0,5]}(C) \wedge \Box\neg(O) \wedge \Box\neg(q_1) \quad [mode : \textit{Steer}]$$

$$\phi_2^3 = \Diamond_{[0,10]}(G) \wedge \Box\neg(O) \wedge \Box\neg(q_1) \quad [mode : \textit{Steer}]$$

$$\phi_2^4 = \Box(G) \wedge \Diamond_{[0,10]}(G)' \quad [mode : \textit{Grasp}]$$

$$\phi_2^5 = \Diamond_{[0,10]}(H_2) \wedge \Box\neg(O) \wedge \Box\neg(q_1) \quad [mode : \textit{Steer}]$$

$$\phi_2^6 = \Box(H_2) \quad [mode : \textit{Land}]$$

Notice, that the constraint $\Box\neg(q_j)$, where $j \in \mathcal{N}_i(t)$, $i \in \{1, 2\}$, enforces the quadrotors to avoid

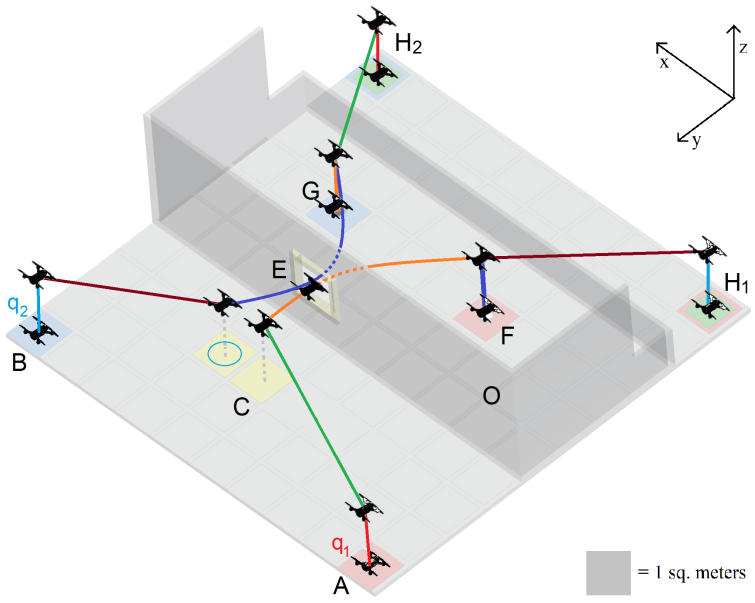


Figure 2.4: The resultant composed trajectories for the sub-tasks for $q_1$ and $q_2$ operating simultaneously.

collision when within $\rho$ proximity of its neighbors. In practice, the UAV that is the first to reach region $C$ and is closest to the window $E$, gets to go through first. The other UAV has to wait in the default *Hover* mode at $C$.

Each UAV sequentially solves the set of MILPs for its respective sub-task specifications, and moves along the generated optimal trajectory for the corresponding sub-task. The final mission trajectory is generated by recursive composition of all these optimal sub-task trajectories. Figure 2.4 shows the resulting composed trajectories for both the quadrotors operating simultan--eously. Both UAVs safely avoid the obstacles and evacuate their respective objects within given finite-time limits. As expected, quadrotor $q_2$ waits at $C$. The number of circular rings at $C$ (see Fig. 2.4) correspond to the waiting time for $q_2$ in terms of discrete time steps.

Table 2.1 provides some insight into the timing analysis of each sub-task for both quadrotors. A closer look at this data indicates that all timing constraints are indeed satisfied. Moreover, the computation time for each sub-task indicates that the proposed method can be implemented

Table 2.1: Timing analysis for the $\phi_i^k$ for $N = 2$

| **Task $\phi_{\mathbf{i}}^{\mathbf{k}}$** | **Computation (sec)** | **Execution (steps)** |
|---|---|---|
| $\phi_1^1$ $(A - A')$ | 2.7 | $2 \leq 5$ |
| $\phi_1^2$ $(A - C)$ | 6.3 | $4 \leq 5$ |
| $\phi_1^3$ $(C - F)$ | 10.3 | $7 \leq 10$ |
| $\phi_1^4$ $(F - F')$ | 3.0 | $3 \leq 10$ |
| $\phi_1^5$ $(F - H_1)$ | 5.7 | $6 \leq 10$ |
| $\phi_1^6$ $(H_1 - H_1')$ | 2.5 | $2 \leq 5$ |
| $\phi_2^1$ $(B - B')$ | 2.7 | $2 \leq 5$ |
| $\phi_2^2$ $(B - C)$ | 5.8 | $4 \leq 5$ |
| $\phi_2^3$ $(C - G)$ | 11.1 | $8 \leq 10$ |
| $\phi_2^4$ $(G - G')$ | 3.0 | $3 \leq 10$ |
| $\phi_2^5$ $(G - H_2)$ | 5.7 | $6 \leq 10$ |
| $\phi_2^6$ $(H_2 - H_2')$ | 2.5 | $2 \leq 5$ |

in real time. To the best of the authors knowledge, these are one of the fastest computation times reported in the existing MTL-based planning literature. The secret to this reduction in computational complexity lies in our divide-and-conquer approach. From an implementation point of view, the performance can be further improved by using hardware which is optimized for computation (such as Nvidia Jetson TX2 etc.).

### 2.5.2 Case Study II: Scalability (N UAVs)

We now consider the case of $N$ number of UAVs to investigate the scalability features of our method. For this case study, we increase the number of UAVs in the simulation setup, one at a time until one of the finite-time constraints in any of the sub-task specifications is violated for at least one of the UAVs. To keep the analysis consistent, we keep the finite timing constraints for all UAVs the same as before.

It turns out that for $N = 7$, the sub-task specification $\phi_7^3$ fails the satisfaction criterion and hence no solution exists for this sub-task. That is, for $N > 6$, one of the UAVs fails to reach its

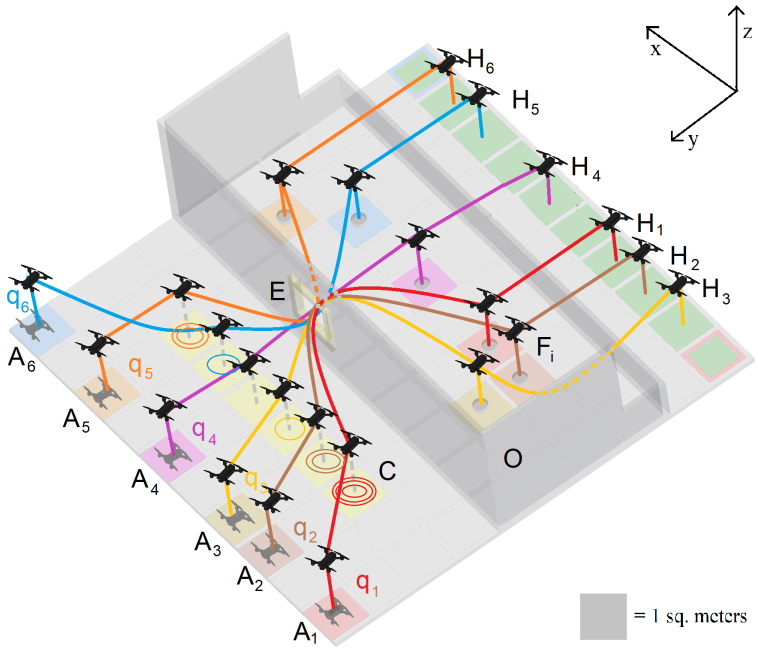


Figure 2.5: The resultant composed trajectories for the sub-tasks for $N = 6$ UAVs operating simultaneously. As before, the number of circular rings at $C$ correspond to the waiting time for the $i^{th}$ quadrotor in terms of discrete steps.

respective target object within the finite-time limit of 10 units. The reason is that beyond $N = 6$, one access point to the environment is not sufficient to meet the specified timing constraints for all the UAVs, since now the quadrotors have to queue up for longer duration at $C$ in order to avoid collisions at $E$.

However, this problem can be solved simply either by increasing the finite-time limits for this sub-task, or by executing the mission in an environment with multiple access points. Another way to look at this limitation is to identify that $N = 6$ is the maximum number of UAVs that can be deployed successfully under these safety and timing constraints in this particular workspace, and adding more UAVs does not provide any additional value in terms of the success of the mission. Therefore, it is more of a limitation of the workspace than our approach itself. Figure 2.5 shows the resulting composed trajectories for 6 UAVs, in which case all safety and

timing constraints are satisfied. The computation and execution times for the sub-tasks for $N = 6$ case, indicating the satisfaction of all timing constraints, can be found in the Table 2.2.

Table 2.2: Timing analysis for the $\phi_i^k$ for $N = 6$

| **Task $\phi_i^k$** | **Computation (sec)** | **Execution (steps)** |
|---|---|---|
| $\phi_1^1\ (A_1 - A_1')$ | 2.7 | $2 \leq 5$ |
| $\phi_1^2\ (A_1 - C)$ | 5.3 | $4 \leq 5$ |
| $\phi_1^3\ (C - F_1)$ | 11.3 | $9 \leq 10$ |
| $\phi_1^4\ (F_1 - F_1')$ | 3.0 | $3 \leq 10$ |
| $\phi_1^5\ (F_1 - H_1)$ | 5.7 | $6 \leq 10$ |
| $\phi_1^6\ (H_1 - H_1')$ | 2.5 | $2 \leq 5$ |
| $\phi_2^1\ (A_2 - A_2')$ | 2.7 | $2 \leq 5$ |
| $\phi_2^2\ (A_2 - C)$ | 5.8 | $4 \leq 5$ |
| $\phi_2^3\ (C - F_2)$ | 10.7 | $8 \leq 10$ |
| $\phi_2^4\ (F_2 - F_2')$ | 3.0 | $3 \leq 10$ |
| $\phi_2^5\ (F_2 - H_2)$ | 5.7 | $6 \leq 10$ |
| $\phi_2^6\ (H_2 - H_2')$ | 2.5 | $2 \leq 5$ |
| $\phi_3^1\ (A_3 - A_3')$ | 2.7 | $2 \leq 5$ |
| $\phi_3^2\ (A_3 - C)$ | 5.8 | $4 \leq 5$ |
| $\phi_3^3\ (C - F_3)$ | 9.3 | $6 \leq 10$ |
| $\phi_3^4\ (F_3 - F_3')$ | 3.0 | $3 \leq 10$ |
| $\phi_3^5\ (F_3 - H_3)$ | 7.7 | $8 \leq 10$ |
| $\phi_3^6\ (H_3 - H_3')$ | 2.5 | $2 \leq 5$ |
| $\phi_4^1\ (A_4 - A_4')$ | 2.5 | $2 \leq 5$ |
| $\phi_4^2\ (A_4 - C)$ | 5.8 | $3 \leq 5$ |
| $\phi_4^3\ (C - F_4)$ | 7.9 | $5 \leq 10$ |
| $\phi_4^4\ (F_4 - F_4')$ | 3.0 | $3 \leq 10$ |
| $\phi_4^5\ (F_4 - H_4)$ | 5.5 | $6 \leq 10$ |
| $\phi_4^6\ (H_4 - H_4')$ | 2.5 | $2 \leq 5$ |
| $\phi_5^1\ (A_5 - A_5')$ | 2.7 | $2 \leq 5$ |
| $\phi_5^2\ (A_5 - C)$ | 5.8 | $3 \leq 5$ |
| $\phi_5^3\ (C - F_5)$ | 10.1 | $7 \leq 10$ |
| $\phi_5^4\ (F_5 - F_5')$ | 3.0 | $3 \leq 10$ |
| $\phi_5^5\ (F_5 - H_5)$ | 5.7 | $6 \leq 10$ |
| $\phi_5^6\ (H_5 - H_5')$ | 2.5 | $2 \leq 5$ |
| $\phi_6^1\ (A_6 - A_6')$ | 2.7 | $2 \leq 5$ |
| $\phi_6^2\ (A_6 - C)$ | 6.3 | $5 \leq 5$ |
| $\phi_6^3\ (C - F_6)$ | 9.5 | $6 \leq 10$ |
| $\phi_6^4\ (F_6 - F_6')$ | 3.0 | $3 \leq 10$ |
| $\phi_6^5\ (F_6 - H_6)$ | 5.7 | $7 \leq 10$ |
| $\phi_6^6\ (H_6 - H_6')$ | 2.5 | $2 \leq 5$ |

## 2.6 Chapter Summary

In this chapter, we have proposed a hybrid compositional approach to rescue mission planning for quadrotors with MTL specifications, and have presented an optimization-based method which can be implemented in real time. Using a simple yet realistic search and rescue test case, we have demonstrated the computational efficiency of our approach, and have shown that by breaking down the mission into several sub-tasks, and by using a hybrid model for the system, it is possible to solve the challenging problem of motion planning for multiagent systems with rich dynamics and finite-time constraints in real time.

The original problem for only two quadrotors, if solved directly without our approach, results in roughly 3 times more linear-constraints in the MILP formulation, which in turn increases its computation time to around 9 minutes in total for each UAV. Our approach clearly results in a massive reduction is computational complexity of the problem by increasing the efficiency of its solution.

In addition to some promising results, this work also poses many new and interesting questions as well. For example, given a finite-time constraint for the whole mission, what is the best or optimal way to divide the timing constraints among various sub-tasks. Of course it is a scheduling problem, and is dependent on many factors such as robot dynamics, its maximum attainable speed, and nature of the sub-tasks as well. Here, the individual sub-task timing constraints were constructed in a relaxed and uniform fashion. However, it is worth noticing that using a rich dynamical model for the UAV puts less constraints on its maneuverability, and hence can allow it to tackle more conservative finite-time constraints as well. For example, the *Steer* mode in our model allows the quadrotor to achieve speeds as high as 1.5 m/s, which is not

possible with the usual single mode *Hover* linearization only.

We anticipate that the scalability features of this approach can be shown with even greater number of UAVs in an environment with multiple access points for instance. Detailed performance comparison of this hybrid approach with some decentralized collision avoidance methods will be beneficial as well. Extension of this work with different tasks, dynamic obstacles other than the UAVs themselves, conservative time constraints, and studying the time-robustness properties of MTL can also yield interesting results, and are all great directions for future work.

# Chapter 3: Cooperative Mission Planning for Multiagent Systems with Distri--buted Consensus Dynamics in a Leader-Follower Setting

Search and rescue of survivors in an emergency or a natural disaster is a strenuous and time-critical task. In many cases, a team of robots can be useful as first responders [51], for identifying and locating target entities before sending in humans or other robots for evacuation [52]. This not only saves lives by providing the much needed help, but also makes the entire operation time-efficient and cost-effective [53].

In this chapter, we explore a new planning approach for identifying survivors in a simulated, hazardous and cluttered environment with a team of ground robots. The main idea is to highlight the versatile features of our composable, realtime MTL-based planning framework proposed in Chapter 2, by demonstrating its flexibility to seamlessly incorporate different system dynamics and controllers alike, while still providing guarantees on safety and finite-time mission completion. As a result, we develop a hybrid planning method for a multiagent team of robots, which can cooperate together to accomplish multiple tasks during a single mission. The three tasks selected in our particular use case are rendezvous, navigation, and persistent coverage, in a leader-follower setting. Following are the main contributions of this chapter:

- A hybrid, realtime, and composable framework for planning cooperative multi-task missions using MTL specifications under the assumption of known environment, and using generic

system dynamics and controls.

- A leader-follower mission planning method with safety and finite-time guarantees for the leader and the followers[1].

- A comparative study of how the (communication) graph-connectivity between the leader and the individual followers relates to their performance during the multi-task mission.

Before diving into the details, we first describe our use case, that is a multiagent search and rescue mission in a cluttered environment.

The storyline for this search and rescue operation goes as follows. A team of researchers were performing experiments in a secret facility located in a remote area, when an accident occurred and all communication with the base station was lost (see Fig. 3.1). We are tasked with the deployment of a team of robots to search and flag or identify survivors at the facility so they can be rescued. The agents are deployed via an air drop in the vicinity of the facility. Outside the facility, no communication infrastructure is available, so the agents need to ensure that they somehow remain in contact (i.e., connected) to exchange information while navigating through a cluttered environment.

[1] * The guarantees on followers are *soft* in comparison to the leader by design. More details on this later on in this chapter.


Figure 3.1: Simulation environment: The facility is represented by a red rectangle on top right. Orange polygons represent the survivors whose locations are unknown. A team of robots (shown as blue polygons) is tasked to reach the facility by navigating the cluttered environment safely and flag the survivors in given, finite time.

For security reasons, only one of the senior agents (i.e., the leader) is made privy of the location of the facility. While inside the facility, agents are able to use the existing communication infrastructure to search for the missing researchers. A survivor can be flagged for rescue if an agent is sufficiently close. It is required for this team to safely complete this mission in a given, finite time. To detect obstacles, each agent is equipped with multiple range sensors, uniformly spaced around the body of the robots (i.e., in the directions away from the center of the robot). Each sensor returns the distance to any object detected in the range. If no objects are detected, the sensors return a value of infinity (this is to keep our setup inline with the Robotarium [54], for simulation and implementation needs, which we use to validate our method later in this chapter).

To solve the stated problem, we propose a hybrid decision-making strategy, which is based on a multiagent leader-follower network. For the leader (selected at random), the mission is modeled as a metric temporal logic (MTL) specification, whereas the followers use weighted consensus dynamics to follow the leader. We formulate the mission as a minimum time-optimal control problem, and convert it to a set of readily solvable mixed-integer linear programs (MILPs) using techniques of decomposition and translation for MTL specifications from Chapter 2. Our method comprises of three distinct steps:

- In the first step, a team of robots is deployed as a connected network near the rescue site, and the agents rendezvous in order to establish a fully-connected communication graph using distributed consensus-based dynamics with obstacle avoidance.

- In the second step, using only local sensing information, the agents navigate through the cluttered environment in a leader-follower configuration to safely reach the target facility.

- Finally, in the third step, we model the *searching for survivors* as a persistent coverage problem with centroidal voronoi tessellations (CVT), using a modified version of Lloyd's algorithm [55], with Gaussian density functions.

We demonstrate the efficacy of our method using a Robotarium [54] simulation in MATLAB with five agents. Our proposed approach is hybrid, composable, exhibits realtime performance, and provides inherent guarantees on safety and finite-time mission completion.

## 3.1 Related Work

As discussed earlier, planning search and rescue missions is one of the most time-critical applications for autonomous multiagent systems. For this reason, over the last few decades, it has been a well-studied topic in the planning literature [56, 57]. Both centralized [58], and decentralized [59] approaches have been studied, using teams of unmanned ground [60], and aerial [61, 62] vehicles to facilitate in the search and rescue of survivors in case of a natural disaster, for instance [63]. However, autonomy, mobility, and reliability still remain three of the biggest challenges with designing such multiagent systems and their planning frameworks [64]. A bulk of the discussion on these existing multiagent planning methods can be simply carried over from Chapter 2. However, in addition to the safety of the humans, the robots, and the infrastructure involved, there is still room for more emphasis to be put on the realtime computation needs, and finite-time execution guarantees, for the majority of these methods.

Therefore, in this chapter, with this particular multi-task search and rescue problem, we aim to bring together several additional concepts from the fields of classical control theory and algebraic graph theory. As will be clear from the problem formulation and the solution approach later on, we intend to combine the guarantees of finite-time mission completion using metric temporal logic [44] with several key ideas in distributed control of networks [65]. These ideas include consensus-enabled rendezvous [66], formation control of leader-follower networks [67, 68], safe navigation design with edge-tension functions [55], and persistent area coverage with voronoi tessellations [69, 70]. While there is an abundance of existing works for each of these individual topics, a hybrid strategy of combining them together for an efficient mission planning algorithm with MTL specifications, has not been studied before. Now we have briefly discussed

some key problems with potential functions already in Chapter 2. Voronoi tessellations can be hard to compute and their convergence is usually slow as well, especially in higher dimensions (i.e., >2). However, the purpose of using these methods here in conjunction with the MTL-based planning, is to showcase the versatile capabilities of our proposed framework using a simple yet effective case study.

## 3.2 Notation and Preliminaries

In this section, we describe some essential mathematical notation and preliminaries which we follow throughout, in the rest of this chapter.

### 3.2.1 Robot Dynamics

We consider the following general, point-robot model dynamics.

$$\dot{x}(t) = u(t)$$

where for all time $t$ continuous

- $x(t) \in \mathcal{X} \subseteq \mathbb{R}^n$ is the state vector of the system, which in our case is the position of the robots in $\mathbb{R}^2$
- $x_0 \triangleq x(0) \in \mathcal{X}_0 \subseteq \mathcal{X}$ is the initial condition of the state vector and
- $u(t) \in \mathcal{U} \subset \mathbb{R}^m$ is the set of control inputs which is constrained in the control set $\mathcal{U}$.

For the sake of convenience, we use the shorthand notation i.e., simply $x$ instead of $x(t)$ to

represent the system trajectory, and $u$ instead of $u(t)$ to represent the control inputs, whenever there is no need to indicate their explicit dependence on time $t$.

### 3.2.2 Edge-Tension Functions

In graph theory, edge-tension functions, as the name suggests, represent tension (either attraction or repulsion) between various edges in a connected graph. In case of a connected network consisting of mobile agents, edge-tension functions are useful in representing the graph energy in terms of the interaction between the various agents. These functions are designed to blow up at the undesired behavior of the agents, similar to a control-barrier function [71]. For example, in case of a $\Delta$-disk connectivity graph, an edge-tension function between the $i^{th}$ and $j^{th}$ agents with collision avoidance can be expressed as follows:

$$\mathcal{E}_{ij}(x) = k\left(\frac{\|x_i - x_j\|}{(\Delta - \|x_i - x_j\|)(\|x_i - x_j\| - \delta)}\right)^2$$

where $k$ is a scalar, $x_i$ and $x_j$ are the positions for the $i^{th}$ and $j^{th}$ agents respectively, $\Delta$ is the maximum connectivity range, and $\delta$ is the minimum safety separation for collision avoidance. Notice that this edge-tension function blows up if the graph becomes disconnected or if the agents tend to collide with each other. Notice that the edge tension function above is differentiable by design. We use similar edge-tension functions for inducing weighted consensus-based dynamics for agents in this chapter, where the weights $w_{ij}$ between the $i^{th}$ and $j^{th}$ agents are defined as follows.

$$w_{ij}(z) = \frac{1}{z}\left(\frac{\partial \mathcal{E}_{ij}(z)}{\partial z}\right)$$

where $z = \|x_i - x_j\|$. These weights can then be used for distributed consensus-based control of the leader-follower network. We provide further details and examples on this topic later on in this chapter.

## 3.3 Problem Formulation

As stated before, we use a leader-follower network approach for the multiagent search and rescue mission planning problem at hand. Our goal is to develop a hybrid and realtime algorithm using the MTL-based mission planning for the leader, and a distributed consensus-based control architecture for the followers. Therefore, the MTL specification for the leader agent is given as follows:

$$\phi = \Diamond\Box_{[0,T]}(H) \wedge \Diamond_{[0,T]}\Box_{[0,T]}(x_T) \wedge \Box\neg(O) \wedge \Box\neg x_j$$

where $H$ is the rendezvous location as shown in Fig. 3.1, $O$ represents the obstacles in the environment, $x_j$ represents the location of a follower agent for all $i \neq j$, $x_T$ is the location of the facility which is known to the leader agent only, and $T$ is total time allowed for the mission completion with $T < \infty$. Essentially, the MTL specification $\phi$ asks the leader agent $i$ to complete the whole mission and flag all survivors within the given, finite-time limit $T$.

The planning problem for the leader agent can now be set up as a minimum time-optimal control problem.

**Problem 3.**

$$\begin{aligned} \min_{x_i,t} \quad & \int_0^t 1 \, \mathrm{d}\tau \\ \textit{s.t.} \quad & \dot{x}_i = u_{leader} \\ & x_i \models \phi \end{aligned}$$

where $t \in [0, T]$, $u_{leader}$ is any control input that drives the leader to $x_T$ safely (i.e., while avoiding obstacles), and $x_i \models \phi_i$ represents a satisfiability constraint on the resulting robot trajectory $x_i$ to satisfy the MTL specification $\phi$.

Now, under the assumption that the network stays at least simply-connected at all times, no optimization is needed for the follower agents, and the following dynamics can be used to accomplish the mission for the follower agents.

$$\dot{x}_i = u_{follower}$$

where $u_{follower}$ is any control input that keeps the followers connected to the leader safely (i.e., while avoiding obstacles, and hence drives them to the target facility as well). Notice that this is what we can call a *soft* constraint on the followers, regarding the finite-time limit $T$ for the mission. This is by design, as we want to study the affect of connectivity between the leader and the followers on the performance of various agents during the mission. We will elaborate this point further later on in simulations.

Therefore, in this setup, if we can solve Problem 3 and come up with feasible control laws for the agents, we can inherently guarantee safety and finite-time mission completion using properties of MTL. We now describe our solution approach for the stated problem formulation.

## 3.4 Solution Approach

As stated before, there are three distinct and inherent parts of the stated problem, namely: **deployment**, **navigation**, and **search and rescue**. In our solution approach, first of all, using Theorem 1 from Chapter 2, we decompose the MTL specification $\phi$ for the leader agent into three sub-tasks, each corresponding to one of the three parts of the mission respectively. The resulting sub-task MTL specifications are given as follows.

$$\phi_1 = \Diamond\Box_{[0,T_1]}(H) \wedge \Box\neg(O) \wedge \Box\neg x_j$$

$$\phi_2 = \Diamond_{[0,T_2]}(x_T) \wedge \Box\neg(O) \wedge \Box\neg x_j$$

$$\phi_3 = \Box_{[0,T_3]}(x_T) \wedge \Box\neg(O) \wedge \Box\neg x_j$$

where $T_1$, $T_2$ and $T_3$ are feasible time-limits such that their sum is less than $T$ (See [44] for details). $\phi_1$ models **deployment**, $\phi_2$ models **navigation** and $\phi_3$ models **search and rescue**. Now, instead of solving Problem 3, we are left with three sub-problems, one corresponding to each distinct part of the mission. We now describe the agent dynamics used for solving each of these parts separately. By default, we consider the case where all agents are fully-connected during the mission. We will point out any differences for the simply-connected case as we go along.

### 3.4.1 Deployment

Part one is **deployment**, which can be considered a consensus-enabled rendezvous problem for all the agents landing at the drop zone $H$. The agents are initially deployed in the outskirts of

the search area, and are equipped with short-range communication equipment, and their interact--ions can be modeled using a $\Delta$-disk graph [55]. Assuming that the agents are initially connected, their first objective is to rendezvous and ensure that all agents can communicate with each other while avoiding collisions. We can write the resulting optimization problem for the leader agent as follows:

**Problem 4.**

$$\begin{aligned} \min_{x_i, u_i, t} \quad & \int_0^t 1 \, \mathrm{d}\tau \\ \textit{s.t.} \quad & \dot{x}_i = \sum_{j \in N_i} w_{ij}(x_j - x_i) + u_i \\ & x_i \models \phi_1 \end{aligned}$$

where $t \in [0, T_1]$, $N_i$ is the neighborhood set for the $i^{th}$ agent with state $x_i$, for all $i \neq j$, and $w_{ij}$ are scalar weights designed using the edge-tension function (from Section 3.4.2), for maintaining $\Delta$-disk connectivity between the $i^{th}$ and $j^{th}$ agents and are given as follows:

$$w_{ij} = 10^{-3} \left( \frac{\|x_i - x_j\|^2 - \delta\Delta}{(\Delta - \|x_i - x_j\|)^3 (\|x_i - x_j\| - \delta)^3} \right)$$

Notice that same dynamics also work for the follower agents in the deployment part of the mission i.e.,

$$\dot{x}_i = \sum_{j \in N_i} w_{ij}(x_j - x_i)$$

This phase is accomplished when the graph becomes fully-connected. In the simply-connected case, rendezvous is not required.

### 3.4.2 Navigation

Part two is **navigation**. Upon successful rendezvous (in the fully-connected case only), the agents switch their objective to enter the search area while avoiding debris. To do this task, the agents enter into a leader-follower formation mode. All agents have access to data from eight range sensors located uniformly around the body of the robots (this is to keep our agents consistent with the Robotarium framework for simulation and testing [54]), which will allow them to sense obstacles, but only one agent i.e., the leader (which is chosen at random) has access to the target facility location $x_T$. We model this navigation phase separately as well for the leader and the follower agents. Let $x_{obs}$ denote an obstacle location as inferred locally using the distance sensors on each agent. It is not explicitly known or will be used but we introduce it here just for the sake of useful notation. The planning problem for the leader agent for the navigation phase is given as follows:

**Problem 5.**

$$\begin{aligned} \min_{x_i, u_i, t} \quad & \int_0^t 1 \, \mathrm{d}\tau \\ \textit{s.t.} \quad & \dot{x}_i = \sum_{j \in N_i} (w_{ij}(x_j - x_i) + w_{obs}(x_{obs} - x_i)) + \alpha \tfrac{(x_T - x_i)}{\|x_T - x_i\|} + u_i \\ & x_i \models \phi_2 \end{aligned}$$

where $t \in [0, T_2]$, $\alpha$ is a scalar, and $w_{obs}$ and $w_{ij}$ are weights designed using edge-tension functions for the obstacles and the agents respectively. These edge-tension functions are given as

follows:

$$\mathcal{E}_{obs} = 20^{-3} \frac{\|x_i - x_{obs}\|^{5/2}}{(\|x_i - x_{obs}\| - \delta_{obs})^2}$$

$$\mathcal{E}_{ij} = 50^{-2} \left( \frac{(\|x_i - x_j\| - d_{ij})}{(\Delta - \|x_i - x_j\|)(\|x_i - x_j\| - \delta)} \right)^2$$

where $\delta_{obs} < \delta$, and $d_{ij}$ is the desired edge length (i.e., distance) between the $i^{th}$ and $j^{th}$ agents such that $\delta \leq d_{ij} \leq \Delta$ for all $i$,$j$.

For the follower agents, no optimization is needed, and the following agent dynamics can be used to accomplish the navigation phase.

$$\dot{x}_i = \sum_{j \in N_i} (w_{ij}(x_j - x_i) + w_{obs}(x_{obs} - x_i))$$

This phase is completed when all agents are inside the target facility. As we will see in the experiments, the connectivity and the selection of the leader contribute to how long all agents take to reach the target facility.

### 3.4.3 Search and Rescue

Part three is **search and rescue**. Once at the facility, the agents switch to exploration mode to search for survivors. All agents now have access to the four corner beacons, that denote the boundary of the search space. While inside the facility, the network connectivity is given by a Delaunay graph [72]. Survivors are identified and marked when an agent is sufficiently close. We

model this phase as a persistent coverage control problem using centroidal voronoi tessellations (CVT). We use a modified version of Lloyd's algorithm with Gaussian density functions. This method provides exponential convergence guarantees to CVT for time-varying coverage in a static environment [70].

The coverage problem for the leader agent can be modeled as follows:

**Problem 6.**

$$\begin{aligned} \min_{x_i, u_i, t} \quad & \int_0^t 1 \, \mathrm{d}\tau \\ \textit{s.t.} \quad & \dot{x}_i = K(C_i(x_i, t) - x_i) + \epsilon \tfrac{(x_B - x_i)}{\|x_B - x_i\|} + u_i \\ & x_i \models \phi_3 \end{aligned}$$

where $t \in [0, T_3]$, $K$ is a scalar, $\epsilon > 0$ is a very small positive number, and $x_B$ is one of the four beacon locations in the rescue site, chosen at random after some finite time $t < T_3$. $C_i(x_i, t)$ represents the next centroid location of a voronoi tessellation computed for the $i^{th}$ agent at time $t$, and is given by the following expression:

$$C_i(x_i, t) = \int_{V_i(x)} q \, \frac{f(q,t)}{m_i(x,t)} dq$$

with

$$m_i(x_i, t) = \int_{V_i(x)} f(q,t) dq$$

where $V_i(x)$ is the voronoi cell for the $i^{th}$ agent and $f(q,t)$ is the standard bivariate Gaussian

density function i.e.,

$$f(q,t) = exp\left(\frac{-1}{2}(q_x^2 + q_y^2)\right)$$

where $q_x$ and $q_y$ are obtained using mesh from the four beacon locations of the search area. Similar to the leader, the general dynamics for the follower-agents in this coverage setting look like following:

$$\dot{x}_i = K(C_i(x_i,t) - x_i)$$

This phase and the mission is accomplished when all survivors are flagged or identified for rescue.

### 3.4.4 Final Trajectory Generation

In order to generate the final trajectories, we solve the set of Problems 4-6 using a MILP-based approach presented in Chapter 2. Here we omit the details of this method, but the key idea is that, each MTL sub-task satisfiability constraint i.e., $x_i \models \phi_i$ for $i = 1, 2, 3, ..$, can be translated to a set of mixed-integer linear constraints using convex geometrical arguments (see [44] for details). This converts each of the Problems 4-6, to a readily solvable mixed-integer linear program, with linear cost function and linear constraints, which can be solved recursively and efficiently using a MILP solver. Notice that the worst-case complexity of these MILPs (Problem 4-6 after translation to linear constraints) is still exponential i.e., $O(2^{mT})$, where $m$ is the number of boolean variables needed to represent $x_i \models \phi_i$ as a set of linear constraints (or equivalently the number of halfspaces required to express an MTL formula), and $T$ is the time-horizon. Therefore,

it is logical to consider decomposing the task specification $\phi$ into simpler sub-tasks as before.

The resultant trajectories for the leader, which are obtained by solving these MILPs, are locally optimal for each individual sub-task, and their existence inherently guarantees safety and finite-time completion of the respective sub-tasks. The final trajectory for the complete mission is generated over time, by composing all the individual optimal sub-task trajectories for the leader. The final path is therefore not optimal but suboptimal with respect to the original mission specification $\phi$. However, as we have argued before, despite this loss of global optimality, the advantages achieved in terms of reduction in computational complexity, and improved performa--nce can be far more important (as is shown in [44]). The results for followers are rather interesting as given the hybrid nature of our approach, they are dependent on the graph configurat--ion and connectivity with the leader, as we will see in the following section.

## 3.5 Simulations and Results

We apply the proposed method in the same workspace as shown in Fig. 3.1. We use the Robotarium template for MATLAB to write our controller for the mission, and use the YALMIP-CPLEX optimization solver for the MILPs. Several parameters such as $\Delta$, $\delta$ etc. are set to their default values as for the actual robots in Robotarium [54].

The experiments are run on an Intel NuC hades canyon. It is a portable computer with an Intel core i7 @ 3.7 GHz CPU, an integrated AMD Vega RX64 GPU, and 32 GBs of memory. In our simulations, we use five agents i.e., $N = 5$ which are deployed as a connected network at the start of the mission. The leader is picked at random from the five agents at the beginning of the navigation phase. We also use five survivors, who are randomly placed within the rescue

site. The time-horizon as well as the finite-time limit on the mission is set to be $T = 6$ minutes[2]. For simplicity, this time-limit is uniformly distributed among the three sub-tasks. As described before, we consider two case studies: a fully-connected case, and a simply-connected case with two sub-categories, in regards to the connectivity between the leader and the followers, and its impact on the performance of various agents during the mission.

### 3.5.1 Case Study I: Fully-Connected Network

Figure 3.2 shows various screenshots from a successful simulation run for the complete mission in the fully-connected case. Table 3.1 shows both the computation and execution times of the three sub-tasks for the leader. Notice that our approach leads to realtime computation of safe trajectories, and all finite-time constraints are satisfied for the leader as well as for all the followers. Table 3.2 shows the execution times for the navigation sub-task for the followers. Notice that in the fully-connected case, the maximum difference between the time it takes for the leader and the followers respectively, to accomplish this sub-task is only $0.6$ seconds. As discussed earlier, this difference is expected by design, because of the *soft* nature of timing-constraints imposed on the followers. This results in reducing the complexity of the planning problem at the expense of providing only *soft* finite-time guarantees for the followers.

Table 3.1: Fully-connected: Leader-timing analysis for the sub-tasks $\phi_i$ for $N = 5$

| **Sub-task $\phi_i$** | **Computation (sec)** | **Execution (min)** |
|---|---|---|
| $\phi_1$ (deployment) | 0.7 | $0.1 \leq 2$ |
| $\phi_2$ (navigation) | 3.3 | $1.7 \leq 2$ |
| $\phi_3$ (search and rescue) | 1.7 | $0.2 \leq 2$ |

[2]Notice that it is another strength of using MTL when specifying missions and tasks, that we can use realtime constraints as well. Although, most of the times, it is convenient to discretize them when dealing with hybrid systems, as was the case in Chapter 2.

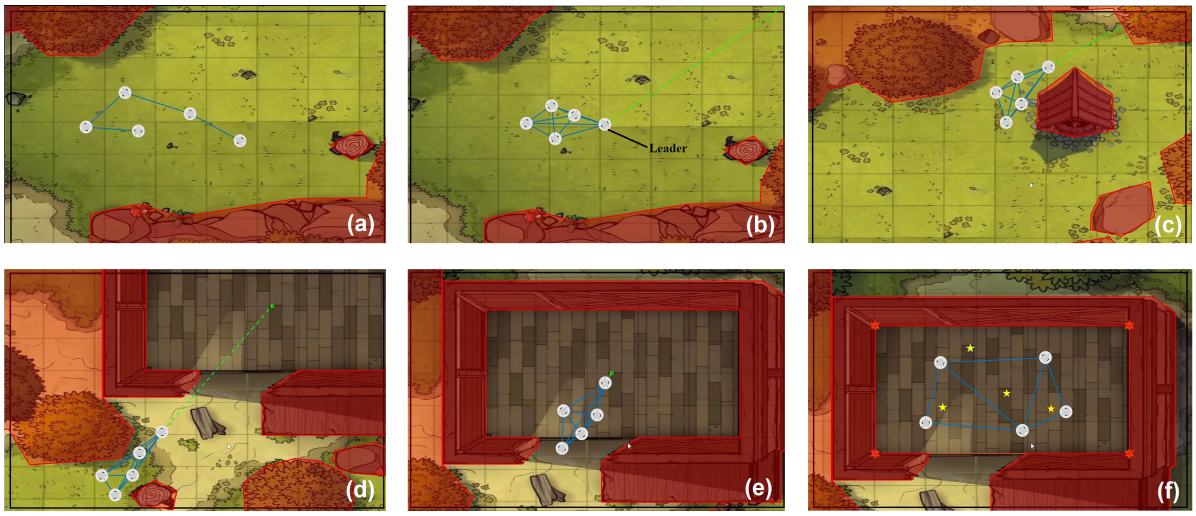


Figure 3.2: Screenshots from a successful simulation run for the fully-connected case, showing: (a) deployment, (b) rendezvous, (c) navigation through cluttered environment, (d) obstacle avoidance while staying fully-connected as a leader-follower network, (e) entering the rescue site, and (f) search and rescue by persistent coverage and identifying the survivors.

Table 3.2: Fully-connected: Followers-timing analysis for the sub-task $\phi_2$ i.e., navigation

| **Follower** | **Execution time (min)** |
|---|---|
| 1 | $1.70 \leq 2$ |
| 2 | $1.71 \leq 2$ |
| 3 | $1.71 \leq 2$ |
| 4 | $1.71 \leq 2$ |

### 3.5.2 Case Study II: Simply-Connected Network

As mentioned before, since the leader is selected at random in the beginning of the mission, it can lead to some interesting network configurations in the simply-connected case. Figure 3.3 shows two such possible network configurations namely *line* and *fork*, which can result from different selection of the leader during the navigation sub-task. Tables 3.3 and 3.4 report the computation and execution times of the three sub-tasks for the leader in both configurations respectively.

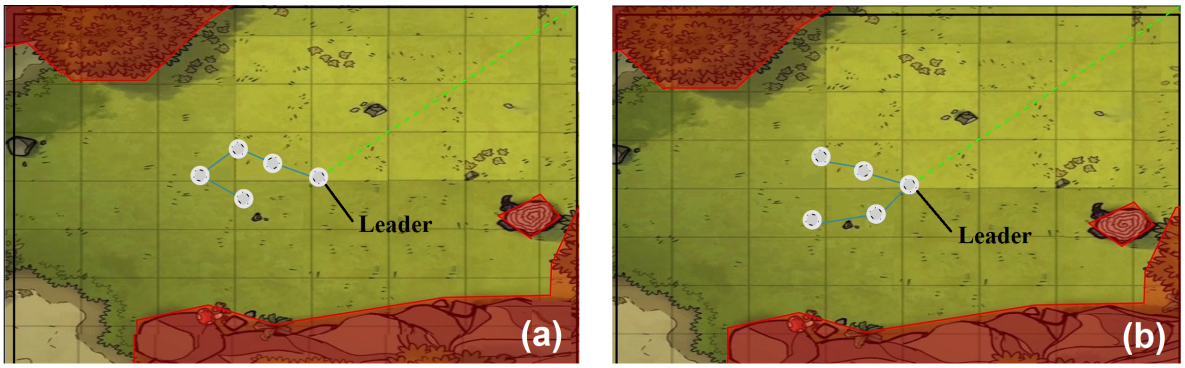


Figure 3.3: Screenshots from a successful simulation run for the simply-connected cases, showing the navigation sub-task for: (a) the line configuration, and (b) the fork configuration.

Table 3.3: Simply-connected (line): Leader-timing analysis for the sub-tasks $\phi_i$ for $N = 5$

| **Sub-task $\phi_i$** | **Computation (sec)** | **Execution (min)** |
|---|---|---|
| $\phi_1$ (deployment) | N/A | N/A |
| $\phi_2$ (navigation) | 3.2 | $1.4 \leq 2$ |
| $\phi_3$ (search and rescue) | 1.8 | $0.2 \leq 2$ |

Table 3.4: Simply-connected (fork): Leader-timing analysis for the sub-tasks $\phi_i$ for $N = 5$

| **Sub-task $\phi_i$** | **Computation (sec)** | **Execution (min)** |
|---|---|---|
| $\phi_1$ (deployment) | N/A | N/A |
| $\phi_2$ (navigation) | 3.3 | $1.6 \leq 2$ |
| $\phi_3$ (search and rescue) | 1.9 | $0.2 \leq 2$ |

The key results for both the simply-connected cases remain the same. That is, the leader and the followers satisfy all the safety and timing constraints for the mission, and are successful in safely navigating to the rescue site, and identify all survivors within given, finite-time limits. However, there is one key difference from the fully-connected case. And that is in the execution times for the leader for the navigation sub-task. Notice that in comparison with the fully-connected case, the execution time for the leader is now smaller for both the simply-connected cases. It is the smallest for the line-configuration of the network, which makes sense, since it is easier for the team of robots to navigate the narrow corridor of the cluttered environment while

moving in a single-file.

Tables 3.5 and 3.6 further show the execution times for the followers in both network configurations respectively. Notice that while the total time of execution for the navigation sub-task has decreased for all agents, the maximum difference between the time it takes for the leader and the followers respectively, to accomplish this sub-task has increased to 3 seconds in the simply-connected case. These results show an interesting trade-off between the different network configurations used, and the resulting performance of different agents in terms of satisfying the finite-time limits of the mission.

Table 3.5: Simply-connected (line): Followers-timing analysis for navigation

| **Follower** | **Execution time (min)** |
|---|---|
| 1 | $1.40 \leq 2$ |
| 2 | $1.41 \leq 2$ |
| 3 | $1.43 \leq 2$ |
| 4 | $1.45 \leq 2$ |

Table 3.6: Simply-connected (fork): Followers-timing analysis for navigation

| **Follower** | **Execution time (min)** |
|---|---|
| 1 | $1.61 \leq 2$ |
| 2 | $1.61 \leq 2$ |
| 3 | $1.62 \leq 2$ |
| 4 | $1.62 \leq 2$ |

## 3.6 Chapter Summary

In this chapter, we developed a hybrid and realtime planning method using the MTL-based mission planning for the leader and a distributed graph-theoretic, networked-control architec--ture for the followers, in order to accomplish a multiagent search and rescue mission. The intention was to enforce the finite-time mission completion guarantees in this leader-follower

network setting using MTL and optimization. Using a relatively simple planning problem as a use case, we have shown some of the strengths of our proposed method. It is hybrid, distributed, composable, exhibits realtime performance, and provides inherent guarantees on safety and finite-time mission completion. While the guarantees for followers are inherently *soft*, the method showcases the flexibility and versatility of our MTL-based planning framework, that it can be seamlessly integrated with different system dynamics and controllers. It also shows that our proposed method can be used in conjunction with other well-known planning methods to produce a hybrid multiagent planner with both centralized and distributed components.

This work also poses many new and interesting questions as well. A plethora of variants of this method can be produced by introducing minor changes. An immediate step moving forward will be to implement this setup with a large number of agents in a simulated environment such as Robotarium. Further options for persistent coverage control methods can also be explored, since CVT-based methods have well-known issues as we have discussed earlier. In addition, instead of merely enforcing finite-time constraints using MTL specifications on the leader with predefined dynamics, we can formulate this problem as an explicit planning problem where we compute the optimal control law instead, as was done in Chapter 2. The upper-bound on finite-time constraints satisfaction for the followers can also be formally specified. It exists in the literature for fully-connected networks [70], but for simply-connected networks it is configuration dependent as well, which makes it a somewhat, cumbersome task. Finally, risk-aware, resilient versions of this method can be studied as well, where instead of a static network of agents, intermittent-disconnections are permitted, where one or more agents can recover from a lost connection, and still continue the mission successfully. All these directions could be good pathways for future work.

# Chapter 4: Composable, Safe, and Realtime Mission Planning for UAV-Based Inspection Tasks

Periodic inspection of safety-critical equipment contributes to a significant percentage of all maintenance tasks in a complex industrial environment. Among these tasks, many are not suitable to be performed by humans for either safety or accessibility concerns, and because of the associated cost, both in terms of time and labor. Modern industry has therefore shown an expanding urgency to replace humans with autonomous robots for many safety and time-critical inspection assignments.

In competition with various other robotic platforms, UAVs have undoubtedly seen a rapid increase in demand for monitoring and inspecting sophisticated equipment, facilities, and infrastr--ucture in recent years. This is partly because of the versatility of the multirotor UAV platforms, which can be easily deployed either indoors or outdoors on land, and even above water. The increasing demand for UAVs is also resulting from a number of relevant applications that they find, in a number of industries. In addition to inspection tasks which we have just pointed out, some other highlights include assistance in disaster relief [73], search and rescue [74], aerial transport and package delivery [75], and enhanced wireless coverage with adhoc UAV networks[76], etc.

Over the past two decades, UAV-based inspection problems in the power sector have gained

massive attention from the experts, for tasks such as high altitude power-line detection and tracking [77], visual inspection of wind-turbine blades [78], and for inspecting large-scale photo--voltaic systems [79]. UAVs have also been used extensively for detecting faults in sizable outdoor structures such as railway tracks [80], pipelines [81], and bridges [82, 83, 84], etc.

It is apparent that UAV-based inspection is a well studied problem in certain outdoor environments where the path planning is rather trivial, and the emphasis is on the visual detection of features in an outdoor structure, rather than the mission planning aspect for UAVs. However, inspecting a complex structure in an industrial workspace is a different challenge altogether. In such cluttered indoor environments, safe and time-critical mission planning becomes a necessity for carrying out UAV-based inspection tasks. Therefore, in this chapter, our focus is not on fault-detection or visual recognition of features on the structure itself. Instead, we consider the problem of inspecting a complicated structure i.e., a pipeline inside an industrial manufacturing facility from a mission planning perspective.

In this chapter, we will utilize some background directly from Chapter 2 because of the significant overlap between the two planning methods for different tasks. Starting with a known environment map, we specify the UAV-based inspection as a metric temporal logic (MTL) specifi--cation and formulate this mission as an optimal control problem with spatio-temporal constraints. We present a systematic way to decompose the mission into several simpler sub-tasks, and consequently represent each sub-problem as a mixed integer linear program (MILP). These MILPs can be solved quite efficiently resulting in an optimal sub-path for the UAV. As shown in Chapter 2, the full path for the mission is generated by composing all the individual optimal sub-paths. Our method can also be extended to using multiple UAVs or other robots for similar inspection tasks. The main contributions of this chapter can be listed as follows:

- A systematic method for representation of inspection tasks in a known environment with MTL specifications.
- An efficient, optimization-based method of composable mission planning for UAV-based inspection tasks in an industrial environment, with guarantees on safety and finite-time mission completion.
- Validation of the proposed method in a simulation environment and discussion on limitations and prospects.

## 4.1 Related Work

We distinguish four families of existing relevant methods. Majority of the discussion of related work in this section is carried over directly from Chapters 2 and 3.

### 4.1.1 Optimal Path Planning Methods

The notion of optimal path planning for robots is well-established in classic motion planning literature [85]. The general idea is to optimize over some cost function and compute a control law for the robot such that it goes from one point to another, while satisfying some constraints [11]. Depending on the cost function, constraints, and system dynamics, several variations of optimal path planning for mobile robots have been historically studied [13]. While majority of these methods consider minimizing the control effort [12], some also consider reaching the goal in minimum time [86]. In this chapter, we aim to minimize the control effort as well, however, we also consider spatio-temporal constraints which are best represented using temporal logic.

### 4.1.2 Linear Temporal Logic-Based Methods

As discussed before, temporal logic [26] can be used to specify complex missions in compact mathematical form. An excess of modern planning literature is based on linear temporal logic (LTL) [27], which is useful for applications where a sequential execution is desired for a number of simple tasks [87], or a team of robots is desired to accomplish multiple tasks in parallel [28]. A MILP-based method using LTL was proposed in [34] and [35] for optimal mission planning using a linear point-robot model. However, LTL, in general cannot incorporate finite-time constraints, and can only guarantee mission completion in the sense of an infinite time-horizon. This makes LTL (including LTLf) not suitable for specifying and planning time-critical missions which require realtime guarantees on their completion.

### 4.1.3 Metric Temporal Logic-Based Methods

Metric temporal logic (MTL) [32] can be used to specify safety and time-critical missions with dynamic task specifications and realtime constraints. A MILP-based optimal mission plann--ing approach using MTL specifications was presented in [36] and [88]. However, in these works, very simple linear dynamics and task specifications were used respectively. Other optimization-based planning methods with MTL specifications for single as well as multiple robots have also been presented in [38, 89]. However, the computation of an optimal trajectory is very expensive (computation time is in the order of 100s of seconds and more). Since the computational complex--ity of an optimization problem with MTL specifications is exponential, it is therefore very important to consider computational tractability of the solution.

### 4.1.4 Composable Temporal Logic-Based Methods

The solution to the computational complexity problem with MTL is to use a divide-and-conquer approach. In some recent works such as [90] and [44], the authors have presented some efficient compositional methods for planning missions with finite-time LTL (or LTLf) and MITL specifications respectively. The former uses composable binary decision diagrams for realtime planning of manipulation tasks for a robotic arm. Whereas in the latter, a fast and hybrid optimization-based approach for rescue mission planning with UAVs is proposed. Our work is closely related to, and can be classified into this particular category of planning problems as well, with the primary focus being the composable, MTL-based planning for UAV-inspection tasks.

## 4.2 Notation and Preliminaries

In this section, we describe some essential notation and preliminaries such as the workspace and the system dynamics for UAV-based inspection tasks. Most of the general notation, for example the MTL syntax etc. is carried over directly from the previous chapters.

### 4.2.1 The Workspace

Figure 4.1 shows the workspace and simulation environment used in this chapter. It is a custom designed CAD model of a smart factory, which is built using Autodesk Inventor Pro. It can be imported and interfaced directly with several simulation software including MATLAB. As shown in Fig. 4.1, the workspace consists of various static and dynamic industrial components such as power units, cooling towers, storage shelves, assembly robots, conveyor belts, and ground carts. Several areas of interest are marked on the map using magenta-colored spots.

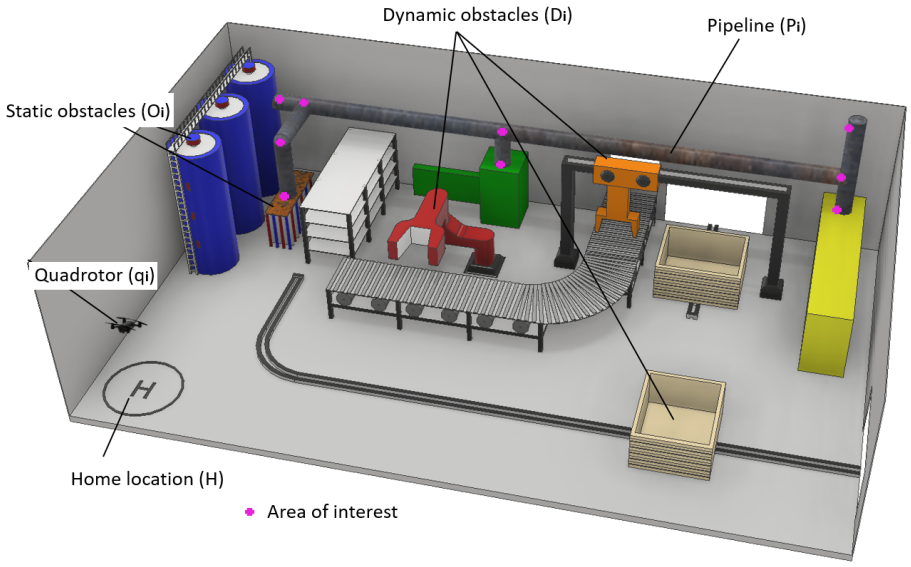


Figure 4.1: CAD model for the workspace used. It is a custom designed simulation environment for a smart factory with several static and dynamic components. The objective for the UAV is to safely inspect the pipeline at the back of the facility within given, finite time.

As will become apparent in our problem formulation later in this chapter, these areas are useful for representing and decomposing MTL specifications for the pipeline-inspection mission. For now, we briefly describe the mission in terms of notation of the workspace. Starting from the home location $H$, the quadrotor $q_1$ needs to visit all points of interest $P_i$ on the pipeline $P$ (where $i \in \{1, 2, ..., N\}$) in order to collect the desired inspection data from its camera. The quadrotor needs to visit all these areas within finite-time limit $T_m$, while avoiding the static obstacles $O$ as well as the dynamic obstacles $D$ at all times. Notice that the pipeline $P$ is also a static obstacle (i.e., $O$) for the quadrotor. Finally, the quadrotor needs to return to home location $H$, and land before it exceeds its flight-time limit $T_f$. Later on, similar to Chapter 2, we also use prime area notation; e.g., $P_i'$ to represent the same 2D area $P_i$. For example, $P_i'$ and $P_i''$ represent two different altitudes (w.r.t. $z$-axis) of the quadrotor while it is inside the same 2D area $P_i$.

### 4.2.2 System Dynamics

System dynamics here are very similar to what we have described earlier in Chapter 2, with the exception of a few extra modes in the hybrid model for the quadrotor.

#### 4.2.2.1 Quadrotor Model

We build a hybrid system model for the quadrotor UAV with seven linear modes, namely *Take off*, *Land*, *Hover*, *Steer*, *Ascend*, *Descend*, and a task specific *Inspect* mode (see Fig. 4.2). As before, the linearization for each mode is carried out separately about a different operating point [42]. This enables our system to have rich dynamics with less maneuverability restrictions, while each mode still being linear. Some modes have the exact same dynamics, and are built in to the hybrid model just for the sake of convenience of implementation. For example, *Take off* and *Ascend* modes have the exact same dynamics with different guard conditions. As in Chapter 2, we will be using discrete-time linear dynamics for each mode in our problem formulation.

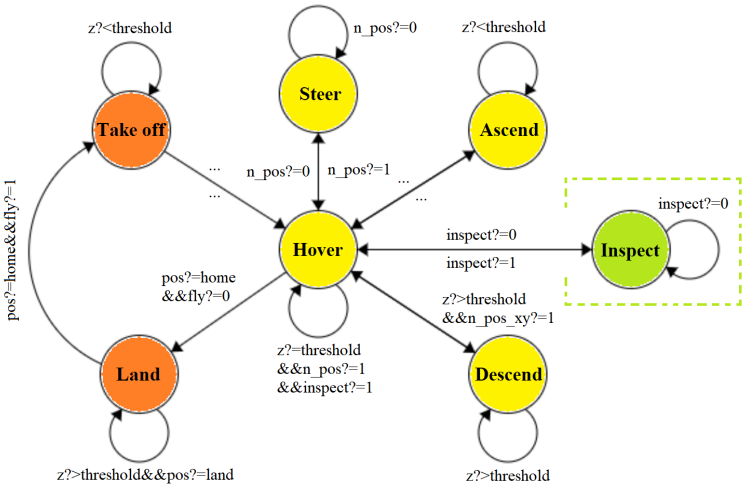


Figure 4.2: The modified hybrid dynamical model for the quadrotor for inspection tasks.

#### 4.2.2.2 The Inspect Mode

Similar to the simplification of *Grasp* mode in Chapter 2, here we represent the *Inspect* mode as a switching combination of *Hover*, *Ascend*, *Descend*, and *Steer* dynamics with special guard conditions (see Fig. 4.3). For this simplified representation of the inspection dynamics, the underlying assumption is that the sensing mechanism i.e., the UAV camera in this particular case, is always facing the object under inspection. This assumption is realistic and is easy to implement in practice, using a 3-axis gimbal-assembly for mounting the camera on to the UAV (as in [74] for example).

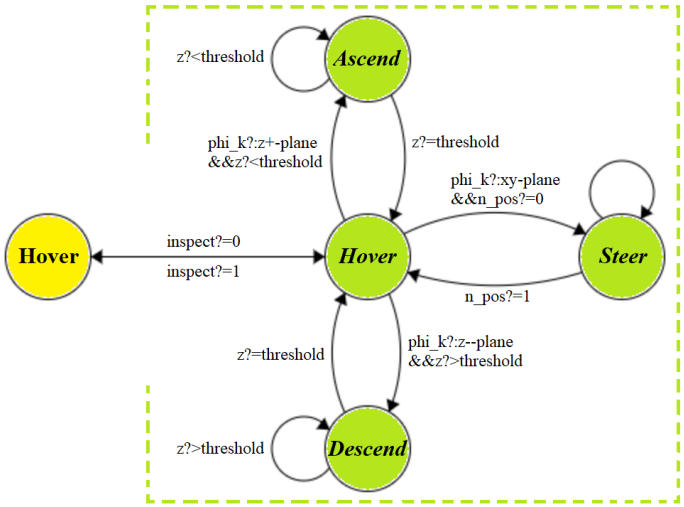


Figure 4.3: The *Inspect* mode expressed as a combination of *Hover*, *Steer*, *Land*, and *Take off* modes (colored green), with special guard conditions.

## 4.3 Method

Given the map of the environment, we come up with a systematic strategy to divide the workspace along one of three axes, to be able to write down a mission specification for the inspection problem. A convenient and rather straight-forward choice is to simply divide the

workspace with three slices along the z-axis as shown in Fig 4.4. This particular sectioning of the workspace is intuitive as well as useful because it suits the geometry of the pipeline and covers most of the points of interest $P_i$ on it.

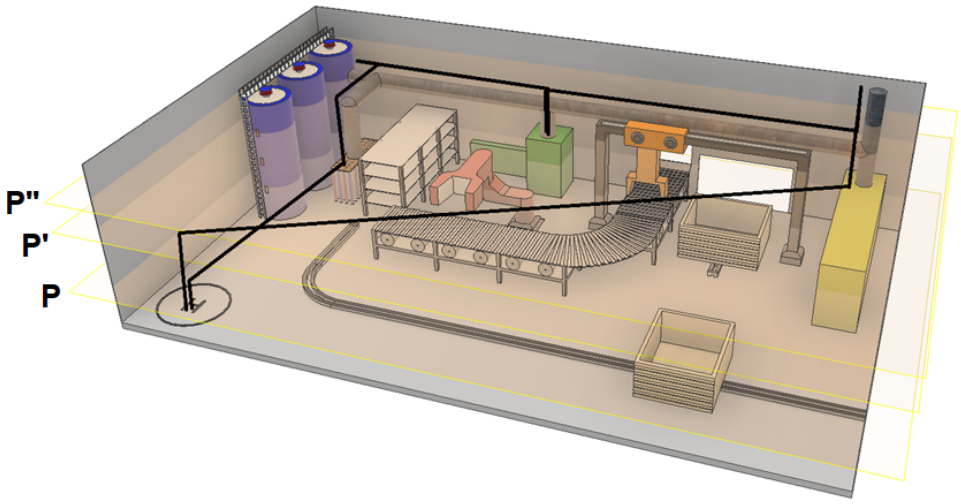


Figure 4.4: The workspace sectioning along the z-axis to model the inspection problem with an MTL specification.

Under this sectioning scheme for the workspace, we can write down a mission specification for the UAV-based inspection task as an MTL formula $\phi$ which is given as follows:

$$\phi = \underset{i}{\wedge}(\Diamond_{[0,T_m]}(P_i) \vee \Diamond_{[0,T_m]}(P_i)' \vee \Diamond_{[0,T_m]}(P_i)'')$$

$$\wedge\Box\neg(O) \wedge \Box\neg(D) \wedge \Diamond_{[0,T_f]}\Box(H)$$

for all $i \in \{1, 2, ..., M\}$ and usually $T_m \leq T_f$; where $P_i$ are the $M$-points on interest on the pipeline (see Fig. 4.1), and the prime notation models the same 2D area of the workspace with an altitude variation of the UAV along the $z$-axis. $T_m$ is the finite-time mission specification in discrete steps and $T_f$ defines the maximum flight-time limit of the UAV. Having specified the mission, we can now formulate this inspection task as an optimal control problem.

### 4.3.1 Problem Formulation

Given the MTL specification $\phi$ for the mission, let $J(x(t, u(t)), u(t))$ be the cost function to be minimized. Then the corresponding optimal control problem for the UAV-based inspection task for quadrotor $q_1$, is given by:

**Problem 7.**

$$\begin{aligned} \min_{x,u} \quad & J(x(t, u(t)), u(t)) \\ \textit{s.t.} \quad & x(t+1) = A(t)x(t) + B(t)u(t) \\ & \mathbf{x}_{t_0} \models \phi \\ & x(t) \in \mathcal{X}_c \end{aligned}$$

Problem 7 is a discrete-time optimal control problem with linear dynamics, which includes a complex MTL satisfiability constraint $\mathbf{x}_{t_0} \models \phi$, as well an optional coverage constraint i.e., $x(t) \in \mathcal{X}_c$. These coverage constraint sets $\mathcal{X}_c$ can be useful in restricting the UAV to a particular set of trajectories to improve visual coverage, depending on the geometry of the structure under inspection. For example, as we will see in the simulation results, in our use case, it can be useful to restrict the UAV to a 3D-spiral or helix-like trajectories around the pipeline to improve the overall inspection coverage.

Now, when it comes to inspection tasks, there could be several additional constraints put in place depending on the object under inspection, the environment around it, the objective of the mission, or depending on what kind of data and post-processing is desired afterwards. For example, one constraint could be to prioritize certain parts of the map over others. Preference could also be placed on what points the robot visits first, and how long or thorough the inspection has to be done for one particular part of the map, and so on. These priority-based, and scene-

awareness constraints are easy to model within the MTL framework, especially with our composit--ional approach. It boils down to the order of scheduling a particular sub-task, and setting its finite-time limit accordingly.

Another interesting constraint for instance, could be a scene-brightness invariance constraint which takes into account the illumination changes in the environment and tries to minimize its effect on the inspection data collected by the UAV sensor i.e., the camera. We can model this constraint using the standard image-brightness equation for a pin-hole camera [91]. The linearized model for image brightness can be stated as follows:

$$\beta(t) = k_{FOV} + \lambda(t) + \omega(t) + v(t)$$

where $\beta(t)$ is logarithm of the image-brightness at time $t$, $k_{FOV}$ is a constant depending on the field-of-view (FOV) of the camera, $\lambda(t)$ is logarithm of the scene illumination at time $t$, $\omega(t)$ is logarithm of the reflectivity of the object under inspection at time $t$, and $v(t)$ represents the camera controls namely the sum of aperture area, shutter speed, and the ISO sensitivity at time $t$. Here, the goal is to optimize the camera controls $v(t)$ such that over time, the change in image-brightness i.e., $\beta(t)$ is kept to a minimum (ideally zero). In this chapter, we do not use any scene-aware constraints in our formulation, since it is difficult to model cameras and sensors in low-fidelity simulation environments like the one used for our simulations. But it can be a useful way to make the mission planning approach more perception-aware, if a high-fidelity simulation or a real-world implementation is desired.

We now describe our solution approach to Problem 7, using the composable, MTL-based planning method presented in Chapter 2.

### 4.3.2 Summary of the Solution Approach

The rest of the steps in the formulation and the solution of Problem 7 follow closely the solution approach described in Chapter 2. We first set up the described inspection mission as a standard optimal control problem in discrete time. Then, we decompose this optimization problem into $M$-sub problems, each corresponding to one sub-task of the original inspection mission. In the next step, we translate the MTL specifications for each sub-task into linear constraints (see Chapter 2).

By decomposing the mission specification $\phi$, and by using the MTL to linear constraints translation mechanism, we replace Problem 7 with a collection of smaller optimization problems, each with a sub-task specification represented as an MTL formula $\phi^i$, and an associated linear mode of the hybrid model.

Here, the linear cost function of choice is again $\sum_{t=0}^{T} |u(t)|$, where $T$ is the discrete time-horizon for the optimal trajectory. Thus, our final formulation of the problem is given by:

**Problem 8.**

$$\begin{aligned} \min_{x,u} \quad & \sum_{t=0}^{T} |u(t)| \\ \textit{s.t.} \quad & x(t+1) = A_l(t)x(t) + B_l(t)u(t) \\ & \mathbf{x}_{t_0} \models \phi^i \\ & x(t) \in \mathcal{X}_c \end{aligned}$$

where $\phi^i$ is the MTL specification for the $i^{th}$ sub-task for the UAV, $A_l(t)$, $B_l(t)$ are the linear system matrices for the $l^{th}$ dynamical mode, and $\mathbf{x}_{t_0}$ is the resulting optimal trajectory for the $i^{th}$ sub-task, and $\mathcal{X}_c$ denotes the coverage constraint set, with $i \in \{1, 2, 3, ..., M\}$, and $l \in \{1, 2, 3, ..., 7\}$.

Consequently, we end up with a collection of $M$-MILPs, which can be solved recursively and efficiently using a MILP solver. The resultant trajectories are locally optimal for each individual sub-task, and their existence inherently guarantees safety and finite-time completion of the respective sub-tasks. The final trajectory for the complete inspection mission is generated over time, by composing all the individual optimal sub-task trajectories. The final path is therefore not optimal but suboptimal with respect to the original mission specification $\phi$. However, despite this loss of global optimality, as discussed in results, the advantages achieved are very significant in terms of realtime computation and fast execution.

## 4.4 Simulations and Results

We apply the proposed method for solving the UAV-based inspection problem (i.e., Problem 8) in the same workspace as shown in Fig. 4.1. The experiments are run through YALMIP-CPLEX solver using MATLAB interface on an Intel NuC hades canyon. It is a portable computer with an Intel core i7 @ 3.7 GHz CPU, an integrated AMD Vega RX64 GPU, and 32 GBs of memory. This setup is transferable to a full-size quadrotor as a companion module for onboard computation. We use a discrete time-horizon for simulation as $T = 50$ units. All dynamics are uniformly discretized at a rate of 5 Hz.

### 4.4.1 Case Study I: Validation

For the validation case study using a single UAV, we end up with 16 sub-tasks in total, which suffices to solving 16 MILPs recursively. Some examples of these sub-task specifications are given as follows:

$$\phi^1 = \diamond_{[0,T_1]}(H) \wedge \square\neg(O) \wedge \square\neg(D)$$

$$\phi^2 = \diamond_{[0,T_2]}(P_1) \wedge \square\neg(O) \wedge \square\neg(D)$$

$$\phi^3 = \diamond_{[0,T_3]}(P_1') \wedge \square\neg(O) \wedge \square\neg(D)$$

$$\phi^4 = \diamond_{[0,T_4]}(P_2') \wedge \square\neg(O) \wedge \square\neg(D)$$

$$\phi^5 = \diamond_{[0,T_5]}(P_3') \wedge \square\neg(O) \wedge \square\neg(D)$$

and so on, until the last two sub-tasks:

$$\phi^{15} = \diamond_{[0,T_{15}]}(H') \wedge \square\neg(O) \wedge \square\neg(D)$$

$$\phi^{16} = \diamond_{[0,T_{16}]}\square(H)$$

For this simulation run, we chose the parameters $T_m$ and $T_f$ to be 320 and 350 steps respectively. Similar to Chapter 2, we use a uniform allocation of the finite-time limits between the sub-tasks. The trajectories for the dynamic obstacles $D$ are assumed to be deterministic, and are known at the time of planning the mission. In addition, for the validation case study, we do not use any coverage constraints in the optimization problem. Therefore, a key assumption for this validation run is that one pass over an area of interest is sufficient to collect the desired inspection data (i.e., images) of that area. In practice, this is not a very good assumption to make, since it requires essentially a 360 degrees field-of-view around the pipeline. However, it is certainly possible to achieve with a specialized gimbal arrangement for the camera mount on to the UAV. Therefore, for a proof of concept for our planning algorithm, this assumption is acceptable.

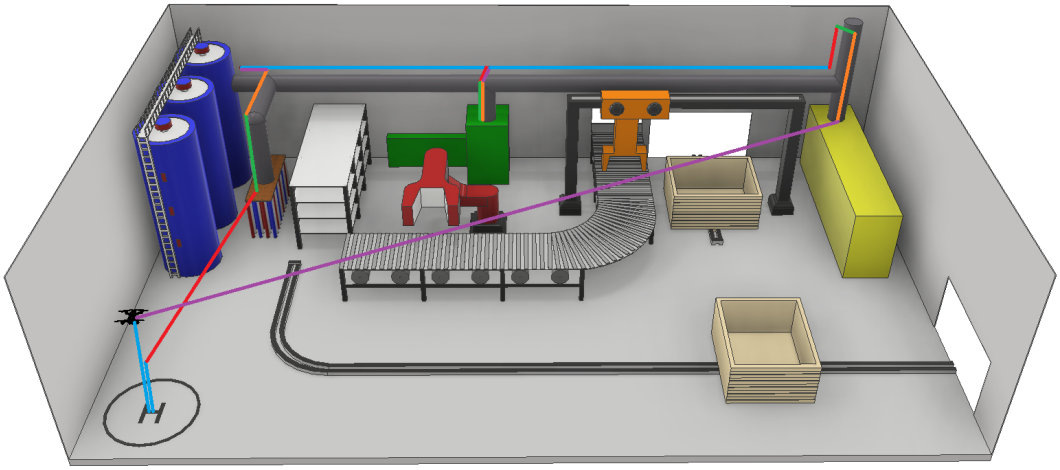

Figure 4.5: The resultant composed trajectories from a successful validation run for all the sub-tasks for quadrotor $q_1$ during the inspection mission.

Figure 4.5 shows the resulting composed trajectories for the quadrotor operating during the whole inspection mission. The computation and execution times for all the sub-tasks are shown in Fig. 4.6. The UAV safely avoids the obstacles and completes the whole mission within the finite-time limits. The minimum and maximum computation times for all individual sub-tasks were observed to be 0.3 and 3.1 seconds respectively, thus achieving close to realtime performance.

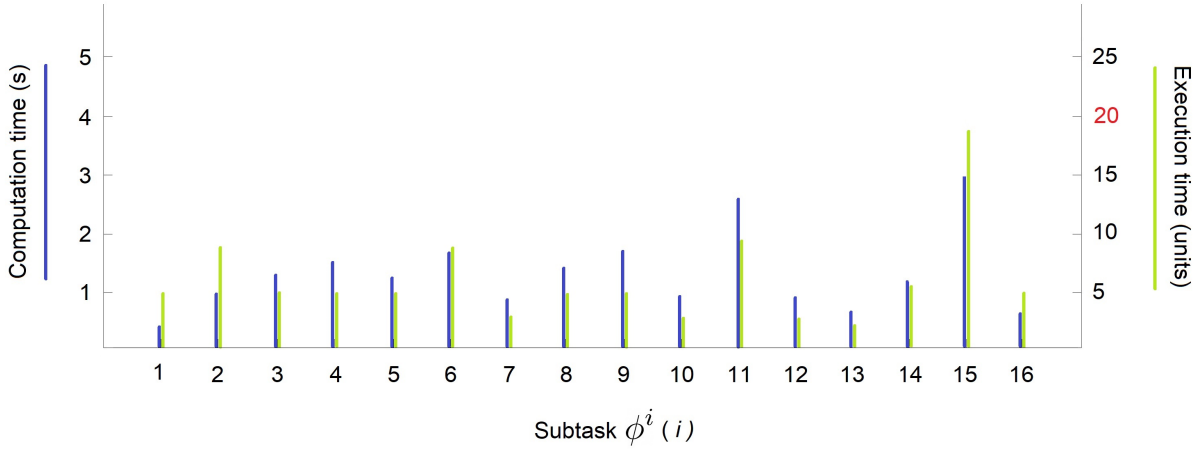


Figure 4.6: The computation and execution times for all the sub-tasks of the inspection mission for the validation run. The blue-plot shows the computation time, while the green-plot presents the execution time for each sub-task $\phi^i$ respectively.

### 4.4.2 Case Study II: Coverage

For the second case study, we use a piecewise-parameterization of a helix (or a 3D-spiral) [92], as the coverage constraint set $\mathcal{X}_c$ in the optimization problem (i.e., Problem 8), for more realistic coverage of the pipeline (see Fig. 4.7).

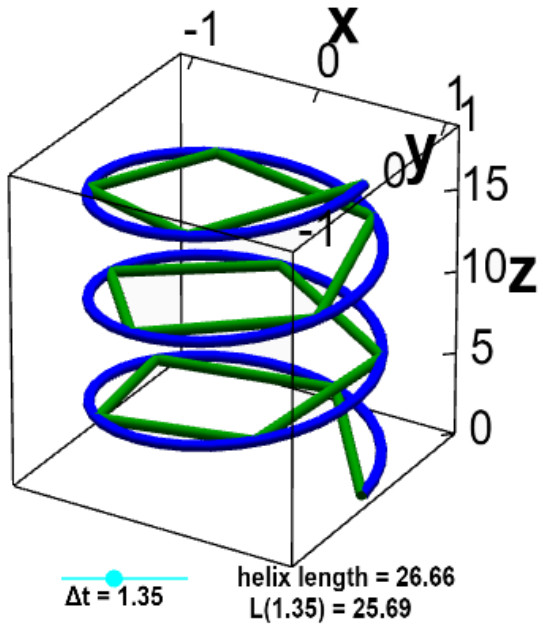


Figure 4.7: Graphic representation of the piecewise parameterization of a helix (or 3D-spiral) used as the coverage constraint set $\mathcal{X}_c$.

For this simulation run, all the sub-task specifications $\phi^i$ remain the same as in the earlier run. However, we update the parameters $T$, $T_m$, and $T_f$ to 60, 480, and 500 steps respectively. These are increased to account for an increase in number of constraints in Problem 8, with the newly added coverage constraints. To keep the results consistent, we use the same uniform allocation of the finite-time limits between the sub-tasks. The trajectories for the dynamic obsta--cles $D$ are assumed to be deterministic, and are known at the time of planning the mission.

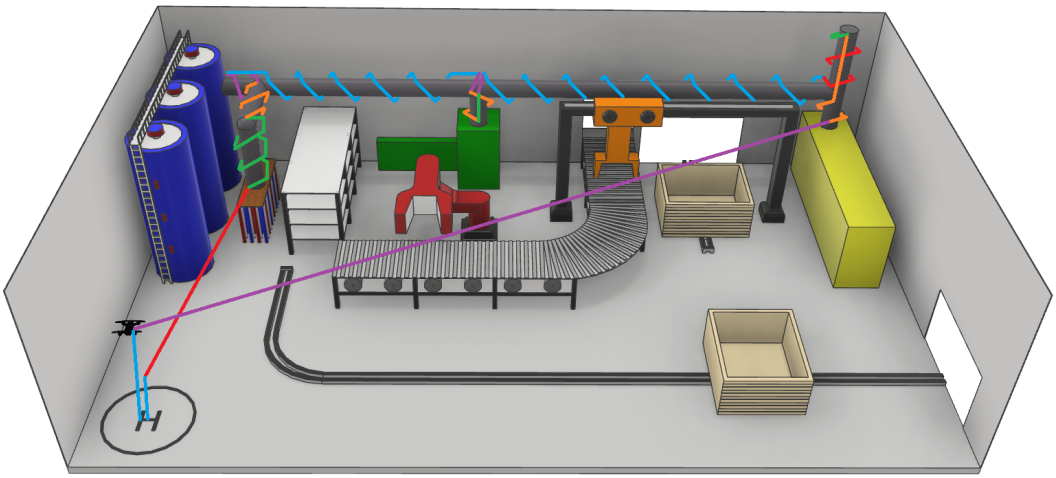

Figure 4.8: The resultant composed trajectories from a successful coverage run for all the sub-tasks for quadrotor $q_1$ during the inspection mission.

Figure 4.8 shows the resulting composed trajectories for the quadrotor operating during the whole inspection mission with coverage constraints. The computation and execution times for all the sub-tasks are shown in Fig. 4.9. The UAV safely avoids the obstacles and completes the whole mission within the finite-time limits. The minimum and maximum computation times for all individual sub-tasks were observed to be 0.3 and 10.9 seconds respectively.

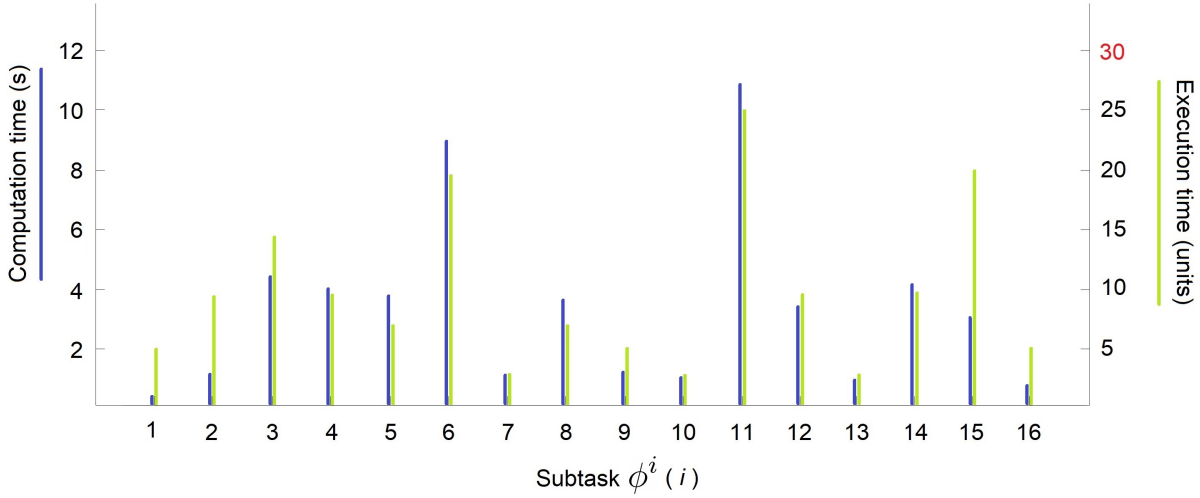


Figure 4.9: The computation and execution times for all the sub-tasks of the inspection mission for the coverage run. The blue-plot shows the computation time, while the green-plot presents the execution time for each sub-task $\phi^i$ respectively.

Notice that the addition of coverage constraints significantly increases the computation and execution times for the UAV for certain sub-tasks, which is expected, since the robot now needs to tackle a lot more obstacles along the way, than before. However, we are still able to achieve close to realtime performance, even in the presence of coverage constraints due to the recursive computation of the optimal sub-task trajectories [44]. To provide a quick comparison of how our method is significantly better than the general TL-based planning methods such as LTLf and MITL-based optimal planners (e.g., [37]): Solving Problem 7 directly i.e., without using our hybrid approach, leads to computation times of approximately 5.5 minutes and 7.2 minutes for LTLf and MITL-based optimal planners respectively. Additionally, note that these computation times are merely for the validation case study i.e., without any special coverage constraints.

## 4.5 Chapter Summary

In this chapter, we have proposed a hybrid, compositional approach to mission planning for UAV-based inspection tasks using MTL specifications, and have presented an optimization-based method which can be implemented in almost real time. Using a simple but relevant use case for modern industry, we have demonstrated the computational efficiency of our approach, and have shown that by breaking down the mission into several sub-tasks, and by using a hybrid model for the system, it is possible to solve the challenging problem of mission planning in cluttered environments, with relatively rich system dynamics and realtime constraints.

An important thing to notice in our case studies is that so far we have used only one UAV under the assumption that $T_m \leq T_f$. This condition does not need to be satisfied all the time because of the composable nature of the proposed approach. Therefore, it is possible to design

a simple scheduling scheme for a multi-UAV inspection mission as well, where we have $T_m \leq T_{f1} + T_{f2} + ... + T_{fN}$, for $N$ number of UAVs for instance. The UAVs can then be deployed either sequentially until each runs out of their flight-time respectively, or they can be deployed in parallel, to inspect different parts of the map simultaneously, with additional collision-avoidance constraints, such as the ones presented in Chapters 2 and 3.

We would also like to point out some of the limitations of the work we have presented in this chapter. As a proof of concept for extending our composable MTL-based planning framework to inspection-oriented tasks, it is indeed a good start. However, there is a lot more that can be done here. The proposed method may not be a good fit for planning inspection tasks in an unstructured outdoor environment for instance, since it requires some workspace information to begin with. This prerequisite of a know environment map is however valid for most of the indoor and structured environments like a smart factory or a power plant etc., since majority of the components there are either static or their behavior is usually well-understood.

The knowledge of the geometry of an inspection surface is also useful, since we can introduce different 3D-coverage constraints in the formulation to improve the overall quality of the inspection mission. This quality of inspection metric can also be quantified in the objective function. Scalability features of this approach for inspection of multiple objects with multiple robots in the same environment needs to be studied as well. Currently, we are looking at all these extensions of this work, and we expect to see more promising results in future.

# Chapter 5: Safe Learning: Self-Monitoring and Self-Correction

Due to uncertainty in the environment or the system (i.e., the robot) itself, the spatio-temporal guarantees obtained at design-time may not correspond to system behavior at runtime. As is emphasized throughout this thesis, for autonomous systems operating in dynamic environm--ents, safety in terms of both space and time are critical requirements for assured autonomy. Therefore, in this chapter, we propose an online adaptation mechanism for safe execution of a complex mission by an autonomous multiagent system. We introduce the ideas of self-monitoring and self-correction for agents using hybrid automata theory and event-triggered model predictive control (MPC). In this setting, we propose a formal, composable notion of safety and adaptation for autonomous multiagent systems, which we refer to as safe learning. We propose a two-phase approach for our safe learning problem. In the monitoring phase, using the hybrid system model, we build a model monitor to check whether the execution sequence at runtime matches the desired execution sequence, and also a safety monitor to check the runtime safety specifications for the system. In the correction phase, using the difference between the predicted and reference system trajectories at runtime, we propose an event-triggered MPC mechanism to drive the system back to the reference trajectory, so as to maintain the initial guarantees on safety and finite-time mission completion. We demonstrate the realtime performance of our proposed approach on a UAV-based surveillance and search and rescue mission, similar to the one discussed in Chapter 2.

## 5.1 Related Work

Self-monitoring and self-correction refer to the problems in autonomy where the autonom--ous agents monitor their performance, detect deviations from normal or expected behavior, and learn to adjust both the description of their mission/task and their performance online, to maintain the expected behavior and performance [93]. The authors in [94] used differential dynamic logic (DDL) to verify safe obstacle avoidance for autonomous vehicles (AVs) with a dynamic window algorithm. They also proposed the passive-safety and passive-friendly safety properties for the system, which are verified with in a dynamic environment containing both stationary and mobile obstacles. The AVs are modeled as hybrid systems, which fully describe their continuous physical motion as well as their discrete control policies.

Similarly, in [95], the authors proposed a hybrid monitoring framework called ModelPlex, which combines offline verification of cyber-physical systems (CPS) with an online validation mechanism, in order to provide various correctness guarantees for the system at runtime. This method uses theorem-proving with sound-proof rules to synthesize three runtime monitors, i.e., a model monitor, a controller monitor, and a prediction monitor, all from the same hybrid model of the system. The model monitor checks the system execution for deviations from the hybrid model. The controller monitor tests the current controller decisions for compliance with the system model, while the prediction monitor evaluates the worst-case safety impact of the current controller decisions, based on the predictions of a bounded-deviation system model. Monitors like these can be constructed at design-time, and they can operate and hence provide a realtime assessment of the state and performance of a hybrid system.

3-valued linear temporal logic or LTL3, is designed for reasoning about LTL properties

for finite-executions, which has been used for runtime verification of LTL specifications in [96]. These LTL3 specifications can be transformed into a monitor automaton, where the transitions of the states are based on runtime sensory information called guard conditions. If the monitor automaton transitions to some *bad* state during the execution of a mission, it implies that a fault is detected at runtime, and the execution should be stopped, or a corrective measure be applied. Therefore, along the same lines, in this chapter, we present a composable framework for monitoring and correcting such runtime faults, that may arise due to uncertainty during a complex mission, which is being carried out by an autonomous multiagent system of robots or vehicles.

An optimization-based method for STL task-specifications, with runtime monitoring and correction was presented in [97]. However, this method is computationally very expensive, with reported computation times of around 100 minutes for tasks involving only a single agent. An MTL-based reinforcement learning (RL) method, with runtime monitoring and correction was also presented in [98]. In this work, the authors used a monitor-guided, modular-RL algorithm, and a timed-automata-based approach to learn the *bad* states in the monitor automaton. This way the agent can successfully and efficiently avoid such states during the execution of the task. While this method is relatively fast to compute and implement for single-agent MTL-tasks, it is not clear how it can be translated to a multiagent setting for a team of autonomous robots, while still keeping intact, its modular structure. Unfortunately, this approach is also not transferable to optimization-based planning methods for more complex MTL-tasks, which is the primary focus of this thesis. Despite key differences, this chapter closely relates to both these works, since we propose a composable and realtime, self-monitoring and correction mechanism for the optimization-based MTL planning framework.

## 5.2 Preliminaries

Before jumping into the design of monitors, we first discuss some essential notation for hybrid automata and the modified new workspace (in comparison to the one from Chapter 2), which we use to validate our approach, later on in this chapter.

### 5.2.1 Hybrid Automaton

We have already seen multiple hybrid models for UAVs and robots in the earlier chapters. Here, we formally define a hybrid model as an automaton, for a general dynamical system [41].

***Definition* 4.** *A hybrid automaton $\mathcal{H}$ is described by a tuple (Loc, Edge, $\sum$, $X$, $Init$, $Inv$, $Flow$, $Jump$) where the symbols have the following definitions.*

- *Loc is a finite set $l_1, l_2, ..., l_n$ of (control) locations that represent control modes of a hybrid system.*
- *$\sum$ is a finite set of event names or labels.*
- *Edge $\subseteq$ Loc $\times$ $\sum$ $\times$ Loc is a finite set of labeled edges which represents discrete changes of control modes in the hybrid system. These changes are labeled by event names taken from the finite set of labels $\sum$.*
- *$X$ is a finite set $\{x_1, x_2, ..., x_m\}$ of real-valued variables. We write $\dot{X}$ for the set of dotted variables $\{\dot{x}_1, \dot{x}_2, ..., \dot{x}_m\}$ which are used to represent first derivatives of the variables during continuous evolution (inside a mode), and we write $X'$ for the primed variables $\{x'_1, x'_2, ..., x'_m\}$ that are used to represent updates at the conclusion of discrete changes*

*(from one control mode to another).*

- *$Init$, $Inv$, $Flow$ are functions that assign three predicates to each control location. $Init(l)$ is a predicate whose free variables are from $X$ and it states the possible valuations for those variables, when the hybrid system starts from location $l$. $Inv(l)$ is a predicate whose free variables are from $X$ and it constrains the possible valuations for those variables, when the control of the hybrid system is in location $l$. $Flow(l)$ is a predicate whose free variables are from $X \cup \dot{X}$ and it states the possible continuous evolution when the control of the hybrid system is in location $l$.*

- *$Jump$ is a function that assigns to each labeled edge, a predicate whose free variables are from $X \cup X'$. $Jump(e)$ states when the discrete change modeled by $e$ is possible, and what the possible updates of the variables are, when the hybrid system makes this discrete change.*

As we will see later on in this chapter, this formal definition for a hybrid automaton is useful in designing various TL-based monitors for hybrid systems. We will use a similar hybrid model for the UAVs for our case studies in this chapter, as in Chapter 4.

### 5.2.2 The Workspace

In this chapter, we will refer to a simulated, surveillance and search and rescue mission defined on the constrained workspace shown in Fig. 5.1. It is a custom-built CAD environment, designed in Autodesk Inventor Pro, with the intention to validate our safe learning (i.e., self-monitoring and self-correction) mechanism, in conjunction with the composable MTL-based planning framework. The reason for using this modified workspace is to mitigate some of the

physical limitations of the environment, observed in Chapter 2. Therefore, in this workspace, we can easily deploy $N$ number of UAVs simultaneously in a complex multiagent mission, without any problems. Moreover, this modified workspace offers several new obstacles thus making the overall experiments and results more interesting.

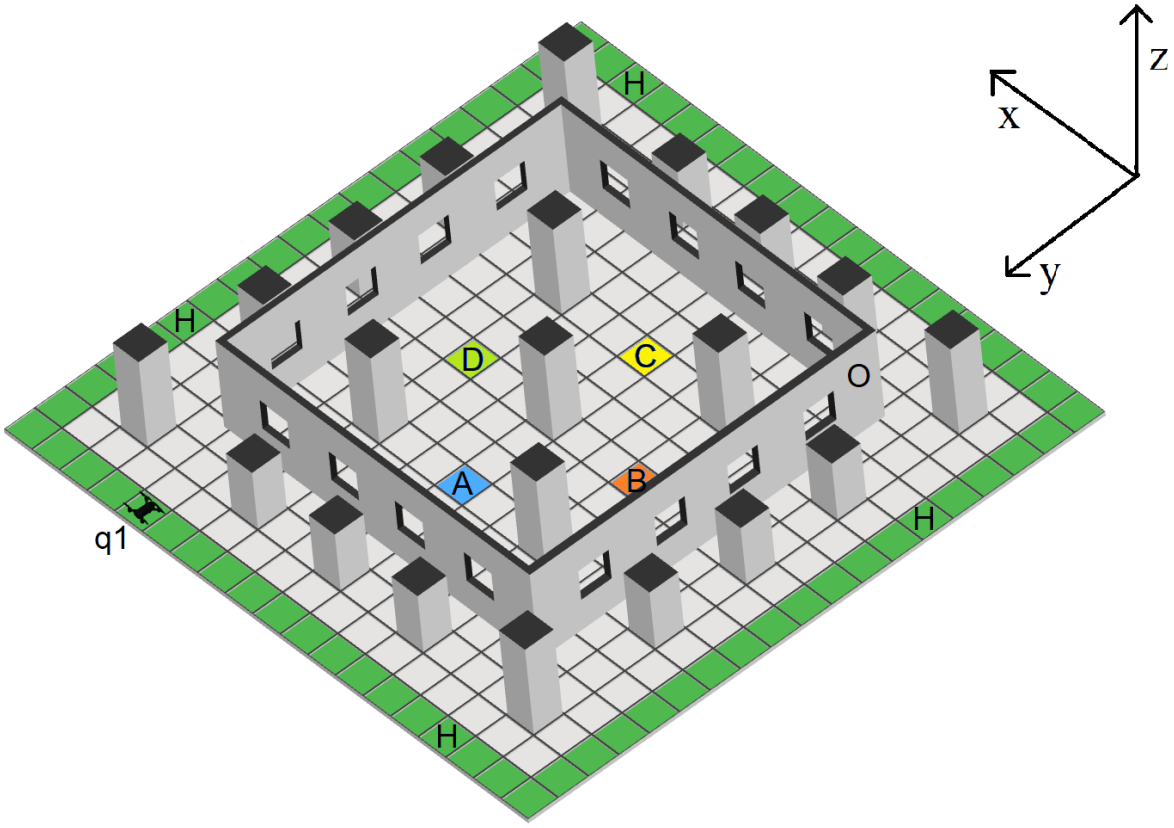


Figure 5.1: CAD model for the modified workspace used. The environment is a 19x19x3 $m^3$ workspace which is divided into several 2D regions of interest that are labeled with alphabets and marked with different colors.

As shown, various areas of interest are marked on the workspace using different alphabets and colors. The role of each of these areas is similar as before in terms of defining the mission with MTL specifications. It will become obvious as we formally define the problems for our case studies later in this chapter. For now, we briefly describe a surveillance mission using a single UAV for our validation case study.

Starting from anywhere on the safe region $H$, a quadrotor UAV $q_1$ needs to periodically visit certain areas of the map located at $A$, $B$, $C$, and $D$ respectively, in a confined environment. These locations are accessible to the UAV only through a number of window openings $E_i$, with dimensions such that they allow only one UAV to pass at a given time. Therefore, if we were to use more than one UAVs for a mission, they will also need to avoid collisions among themselves, in addition to avoiding any obstacles $O$ in the environment. The task for the UAV is to safely visit the respective target areas, and return to the safe zone $H$ in given, finite time. Similar to Chapter 2 and 4, the use of prime region notation, for example $H'$ simply represents an altitude (w.r.t. $z$-axis) variation of the quadrotor while it is in the same 2D region $H$.

## 5.3 Self-Monitoring

Self-monitoring (also referred to as runtime monitoring) can mitigate the problem of reality gap between a model and the real system, especially when used in conjunction with the offline-planning and verification-routines for CPS. Given that autonomous robots and vehicles are cyber-physical in nature, and that any malfunctions can have serious safety consequences, monitoring the system behavior at runtime is critical for their safe operation. In this section, we propose a two-phase process for our self-monitoring problem. In the first phase, we model the robot as a hybrid system, and build a model monitor to check whether the execution sequence for the system at runtime matches the desired execution sequence. In the second phase, we propose a safety monitor design for MTL sub-tasks to check the runtime safety specifications for the system.

### 5.3.1 Model Monitor

Model monitors can be thought of as logical observers, which are useful for monitoring the runtime system execution for robots during a mission. There is a whole area of research devoted to dealing with errors detected by model monitors for CPS [99], which focuses on designing resilient and risk-aware controllers [100], which can deal with such system failures autonomously, during the execution of a mission [101]. In this thesis, however, we do not address such corrective action-design for the individual system failures, and are only interested in detecting such faults to prevent accidents or collisions between agents in a multiagent setting.

In order to design the model monitor, we need to construct a hybrid system model for the robots, where the transitions are based on sensory information called guard conditions. For instance, consider the hybrid system model for a quadrotor UAV in Fig. 5.2. The construction of the model monitor is then straight forward. In each state $s_i$ of the hybrid automaton $\mathcal{H}$, we simply test the previous state $s_{i-1}$, and check whether the transition follows the desired system-trace, as planned in the design phase of the mission. If an error is detected, the execution is stopped.

For example, a desired execution trace for the system model in Fig. 5.2 is given as follows:

$$Take\ off \rightarrow Hover \rightarrow Steer \rightarrow Hover \rightarrow Ascend \rightarrow Hover$$
$$\rightarrow Steer \rightarrow Hover \rightarrow Descend \rightarrow Hover \rightarrow Land$$

Figure 5.3 shows a two-state hybrid model monitor for monitoring this system execution. Since there are no problems with the transitions of this system trace, the model monitor will not go to the *Bad* state, and hence no error is detected.

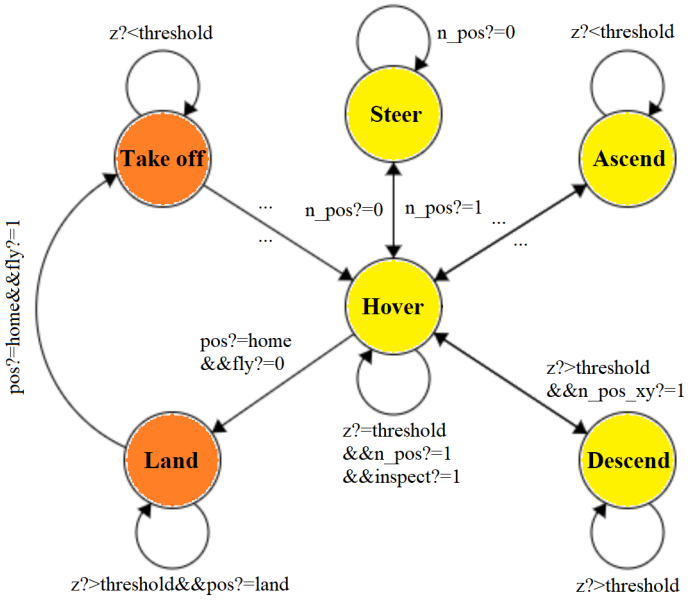


Figure 5.2: A simplified hybrid model for a quadrotor UAV. Some guard conditions are hidden for readability. As before, we use linearized dynamics around different operating points for each mode (see Chapter 2). This makes the model rich in dynamics while still being linear. Notice that it is identical to the hybrid model used in Chapter 4, except for the absence of *inspect* mode, which was specific to inspection tasks only.

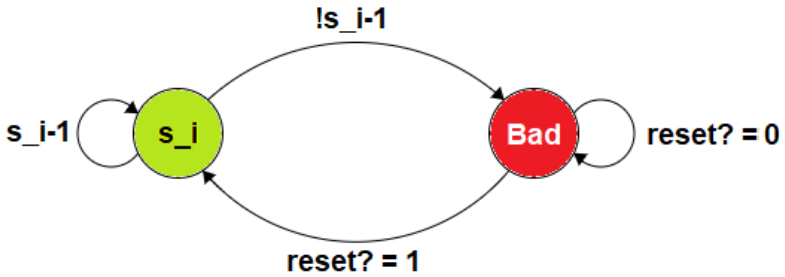


Figure 5.3: A two-state hybrid model monitor for monitoring the system execution at runtime.

Now consider a scenario, where some internal fault within the quadrotor (for example, a broken propeller or some other sensor failure), results in the following system trace:

$$Take\ off \rightarrow Hover \rightarrow Ascend \rightarrow Descend$$

In this case, immediately after the third transition of the system, the model monitor will detect an error and go to the *Bad* state. This is because the system is not supposed to transition from the *Ascend* mode directly to the *Descend* mode, without passing through the *Hover* mode. As soon as the model monitor goes to the *Bad* state, it implies that the system is not behaving as expected and the execution of the task must be stopped. In our case, we simply give a *Land* command to a UAV, if the model monitor detects an error during the execution of a task.

### 5.3.2 Safety Monitor

Safety or task monitors (as the name suggests) are used for monitoring the system execution with respect to the satisfaction of logical and realtime properties, which are represented by a complex mission specification. For designing a safety monitor, instead of specifying and verifying the entire mission specification, we only extract and specify the safety properties that the system must exhibit for a particular sub-task. As we will see in later parts of this chapter, we can also specify additional runtime specifications in safety monitors, which are represented by the MTL sub-task specifications for the mission. However, we first need to go through some formal steps in order to be able to design a safety monitor for MTL sub-tasks.

We start with the LTL specifications first, and gradually build our way up to MTL sub-task monitors. As discussed before, LTL semantics are defined over infinite traces or execution

sequences of a system. In order to formalize the runtime satisfaction of LTL specifications for finite system traces, the authors in [96] proposed new semantics for 3-valued LTL or LTL3, where an LTL formula can evaluate to three different values $\{\top, \bot, ?\}$, instead of the traditional two. The value "?" suggests that it is not possible to conclude a satisfaction or violation of an LTL specification, given the current, finite system trace. We denote the set of all finite words in the execution sequences over $\sum$ by $\sum^*$, and the set of all infinite words by $\sum^\omega$. For any finite words $u$ and $w$, the expression $u \cdot w$ represents the concatenation sequence.

***Definition* 5.** *(LTL3 semantics) Let $\alpha \in \sum^*$ be a finite system trace. The valuation of an LTL3 formula $\phi$ with respect to $\alpha$, denoted by $\alpha \models \phi$, is defined as follows:*

$$\alpha \models \phi = \begin{cases} \top \; if \; \forall \omega \in \sum^\omega : \alpha \cdot \omega \models \phi \\ \bot \; if \; \forall \omega \in \sum^\omega : \alpha \cdot \omega \not\models \phi \\ ? \; o.w. \end{cases}$$

Note that the expression $\alpha \models \phi$ in LTL3 semantics is defined over finite words of the system trace, as opposed to the same expression in LTL semantics, which is defined over the sequences of infinite words. For example, given a finite system trace $\sigma = a_0 a_1 ... a_n$, the property $\Diamond p$ holds true if $a_i \models p$, for some $i$, $0 \leq i \leq n$. Otherwise, the said property evaluates to "?".

LTL3 specifications can be transformed into a monitor automaton where the transitions between the states are based on some guard conditions [96]. If the monitor automaton transitions to some *Bad* state, we should stop the execution or perform some correction maneuver. For instance, in the aerial grasping and rescue with UAVs example in Chapter 2, the UAV should not proceed to the next sub-task in the mission, unless a successful grasping confirmation of the target

object is received. During runtime, a safety (i.e., a sub-task) monitor will detect this problem (if applicable), by transitioning to a *Bad* state. We now formally define an LTL3 task monitor.

***Definition* 6.** *(LTL3 monitor) Let $\phi$ be an LTL formula which takes on the predicates $P$. The monitor automaton $M^\phi$ of $\phi$ is the unique deterministic finite-state automaton (DFA) $M^\phi$ = $(P, Q, q_0, \delta, \lambda)$, where $Q$ is the set of states, $q_0$ is the initial state, $\delta \subseteq Q \times P \times Q$ is the transition relation, and $\lambda$ is a function that maps each state in $Q$ to a value in $\{\top, \bot, ?\}$, such that for any finite trace $\alpha \in \sum^*$, $\alpha \models \phi = \lambda(\delta(q_0, \alpha))$.*

However, there is still a big set of properties for which this LTL3 monitor fails to provide a conclusive evaluation. For example, consider the following LTL specification: $\phi = \Box(a \rightarrow \Diamond b)$ (thanks to [102] for this example), which states that: "once $a$ has been visited, $b$ will always be visited, eventually". For this specification, there does not exist any finite sequence of words which we can use as a good or bad prefix for $\phi$, and therefore, this specification always evaluates to a "?" value. There is a workaround for this problem to some extent, which is done by defining "presumably false" and "presumably true" states, wherever a conclusive evaluation is not possible using LTL3 [103]. However, in our case, this evaluation is still not good enough to be able to monitor finite-time properties of realtime systems (i.e., the MTL specifications). Therefore, we need a few more pieces before we can construct an MTL sub-task monitor.

***Definition* 7.** *(LTL3 sub-task monitor) Given an LTL3 monitor automaton $M^\phi$ = $(P, Q, q_0, \delta, \lambda)$, a sub-task monitor $M^\phi_{LTL3sub}$ is defined as a transition system starting from $q_0$, ending in $q'$ such that $\lambda(q') = \top$, and has the following properties: (1) each edge in $M^\phi$ has been visited at most once, (2) transition to good or neutral state accepts exactly one atomic proposition and all other propositions lead to bad states, and (3) there is only one good state in a given sub-task monitor.*

Given an LTL3 specification, we can generate its corresponding safety monitor automati--cally using the LTL3Tools[1]. From there, the LTL3 sub-task monitors can be constructed using Definition 7. The corresponding MTL sub-task monitors can then be generated by augmenting finite-time constraints according to the atomic propositions and their respective root tasks.

***Definition* 8.** *(Root task) An MTL (more precisely MITL) specification $\phi$ with the structure $\phi = (\phi_1 \vee \phi_2 ... \vee \phi_m) \wedge \phi_s$ is equivalent to $\phi = (\phi_1 \wedge \phi_s) \vee (\phi_1 \wedge \phi_s) ... \vee (\phi_m \wedge \phi_s)$. We denote each $F_i = (\phi_i \wedge \phi_s)$ as the root task of the original specification $\phi$. The MTL specification $\phi$ is satisfied by satisfying any of its root tasks $F_i$.*

It turns out that our decomposition strategy for MTL specifications from Chapter 2 results in sub-tasks that satisfy Definition 8. This is not just a coincidence but comes down to the fact that we decompose the MTL specifications for the mission in such a way that there is only one dynamical mode of the hybrid system associated with each sub-task. By doing so, we ensure that at any given time, it is not possible for a finite system trace $\alpha$ to accomplish two root tasks $F_i$ and $F_j$ simultaneously. Mathematically, it means that at any given time, it is not possible that $\alpha \models F_i$ and $\alpha \models F_j$ for any $i \neq j$. Under this assumption, we can now associate an MTL sub-task monitor to its corresponding root task.

***Definition* 9.** *(MTL sub-task monitor) Given an LTL3 sub-task monitor $M^{\phi}_{LTL3sub}$, and its corresp--onding root task $F_i$, let $\Omega$ denote the set of atomic propositions for root task $F_i$. If $F_i$ contains temporal operators such as $\mathbf{U}_{[t_1,t_2]}p$, $\Diamond_{[t_1,t_2]}p$, or $\Box_{[t_1,t_2]}p$, for $p \in \Omega$, then an MTL sub-task monitor $M^{\phi}_{MTLsub}$ can be constructed by adding a clock constraint $t_1 \leq t \leq t_2$ to the transition that exacts only $p$ on $M^{\phi}_{LTL3sub}$.*

[1] http://ltl3tools.sourceforge.net/

Simply put, using Definitions 5-9, we can construct a safety monitor for each MTL sub-task specification by adding finite-time (clock) constraints to the LTL3 sub-task monitor. Note that in this construction, we only consider atomic propositions which are attached to at most one finite-time (clock) constraint; i.e., we do not consider MTL specifications such as $\Diamond_{[t_1,t_2]}\Box_{[t_3,t_4]}p$ etc. Notice that all MTL specifications, in particular, the sub-task specifications used throughout this thesis, satisfy this assumption.

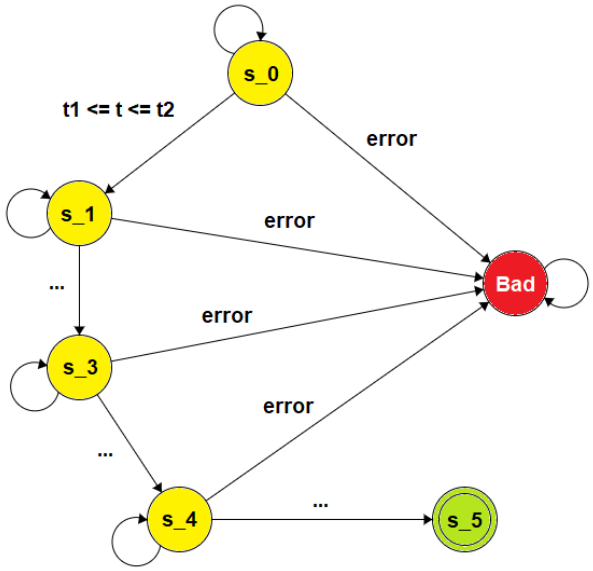


Figure 5.4: Abstract graphical representation of an MTL safety/task monitor as an automaton.

Figure 5.4 shows an abstract graphical representation of an MTL safety monitor. The 4 yellow states are the neutral states with their respective clock-constraints shown on the transitions, the green state is the *Good* or the acceptance state where we want to end up going. The red state (multiple of these are possible) is a *Bad* state. We provide more concrete examples of MTL sub-task monitors later in this chapter when we discuss the validation and performance case studies for a UAV-based surveillance mission. We encode all, whether model or safety monitors in MATLAB using the Stateflow module.

## 5.4 Self-Correction

Once a safety monitor is designed for each sub-task of the mission, it checks for any wrong execution sequences for its corresponding sub-task, and transitions to a *Bad* state of the monitor automaton, if an error is found. This brings us to the next part of this chapter, which deals with self-correction, once an error in the execution sequence of a sub-task is detected.

### 5.4.1 Runtime Monitoring and Correction Criteria

Recall from Chapter 2, that MTL takes on boolean (or binary) predicates but allows finite, realtime constraints in its task specifications. This means that while MTL sub-task monitors can monitor finite-time constraints in real time, they can only provide a binary decision on the satisfaction of a sub-task specification by the system trajectory, at any point during the execution. Therefore, unlike STL-based planning methods [97], when designing a self-correction mechanism with MTL sub-tasks, we can only talk about realtime tolerance and robustness in terms of time but not in terms of space. In some recent works, such as in [104], the authors studied the time-robustness properties of STL tasks. Many of these properties for STL also hold for MTL, since the difference between the two only lies in the kind of predicates they can take on. STL allows real predicates as well as realtime constraints in its task specifications, while as discussed before, MTL only takes on boolean predicates. However, this limitation of the MTL semantics does not hinder us from using MTL sub-task monitors to design a self-correction mechanism using a time-tolerance criterion. The reason why this works is that the space and time-robustness properties of STL and MTL tasks are not independent of each other [105]. To understand this idea, consider this simple example where a robot needs to go from location $A$

to location $B$ in some finite-time limit, as specified by an MTL specification. Note that if the robot somehow deviates from its desired path (in terms of space), it is also likely to violate its finite-time limits for this task, since it is going to take a path that is longer than the optimal one. Using similar arguments, we now design a runtime monitoring and correction criteria for MTL sub-tasks, using in conjunction, a binary space-tolerance condition and a real time-tolerance condition.

Let $T$ denote the time-horizon of trajectory planning for a given MTL sub-task, and let $x_r(t)$ and $u_r(t)$ denote the reference states and the control inputs respectively for $t \in [1, T]$. Notice that $x_r(t)$ and $u_r(t)$ for each sub-task are obtained by solving their respective MILPs (see Chapter 2). During runtime, we define three parameters $\theta_{time(trig)}$, $\theta_{time(max)}$, and $\theta_{space}$, to monitor the execution of each sub-task. $\theta_{time(trig)}$ and $\theta_{time(max)}$ are the triggering and maximum time-tolerances for the sub-task respectively, with $\theta_{time(trig)} \leq \theta_{time(max)}$. $\theta_{space}$ represents the desired space-tolerance which is a binary variable set to 0 by default, and is introduced here only for the sake of notation. We will shortly explain what we mean by the triggering time-tolerance $\theta_{time(trig)}$. At time $t'$, let us denote the observed robot states as $x_o(t)$, where $t \in [1, t']$. Then the predicted states $x_p(t)$ for the robot can be generated using the observed states $x_o(t)$, and the reference inputs $u_r(t)$, until the end of the execution i.e., when $t = T$, as follows:

$$x_p(\tau + 1) = f(x_p(\tau), u_r(\tau)),\ \tau = t', ..., T - 1 \tag{5.1}$$

$$x_p(\tau) = x_o(\tau),\ \tau = 1, ..., t'$$

As usual, the (linear) system dynamics are chosen appropriately for the given sub-task, from the hybrid model $\mathcal{H}$ of the robot. Let $x_p^t$ denote the predicted system trajectory at time $t$. Then we

can evaluate the deviations $r_{time}(x_p^t)$ and $r_{space}(x_p^t)$ for the predicted trajectory. Notice that the space-deviation for the predicted trajectory i.e., $r_{space}(x_p^t)$, just like $\theta_{space}$, is a binary variable which is either 0 or 1. Therefore, realistically, we need a set of conditions on time-tolerances to define an appropriate criteria for triggering the self-correction mechanism. This is where the triggering time-tolerance $\theta_{time(trig)}$, and the time-robustness properties of MTL come in handy. We summarize six possible conditions for runtime monitoring and correction-triggering criteria in Table 5.1.

Table 5.1: Event-triggering criteria for self-correction

| **Condition** | **Result** |
|---|---|
| $r_{time}(x_p^t) \leq \theta_{time(trig)} \leq \theta_{time(max)}$ and $r_{space}(x_p^t) = \theta_{space}$ | No event |
| $r_{time}(x_p^t) \leq \theta_{time(trig)} \leq \theta_{time(max)}$ and $r_{space}(x_p^t) \neq \theta_{space}$ | No event |
| $\theta_{time(trig)} \leq r_{time}(x_p^t) \leq \theta_{time(max)}$ and $r_{space}(x_p^t) = \theta_{space}$ | No event |
| $\theta_{time(trig)} \leq r_{time}(x_p^t) \leq \theta_{time(max)}$ and $r_{space}(x_p^t) \neq \theta_{space}$ | Event |
| $\theta_{time(trig)} \leq \theta_{time(max)} \leq r_{time}(x_p^t)$ and $r_{space}(x_p^t) = \theta_{space}$ | Event |
| $\theta_{time(trig)} \leq \theta_{time(max)} \leq r_{time}(x_p^t)$ and $r_{space}(x_p^t) \neq \theta_{space}$ | Event |

Notice that $\theta_{time(trig)}$ serves as the threshold time-tolerance for the given sub-task after which a correction mechanism is required. A *No event* entry indicates that the execution sequence still satisfies the sub-task specification within the desired tolerance levels, and there is no need for self-correction. When that is the case, we simply use $u_r(t)$ as the control input for the system at time $t$. For the cases which result in an *event*, we need a self-correction mechanism to drive the system back towards the reference trajectory.

### 5.4.2 Event-Triggered Model Predictive Control

We now present an event-triggered MPC scheme for runtime self-correction, where we constantly evaluate if the predicted system trajectory satisfies the sub-task specification. In other

words, whether or not it still maintains a specific space/time-tolerance in comparison with the reference trajectory. The three event-triggering conditions for the MPC module are listed in Table 5.1. At any given time $t$, if any of the three conditions are met, it indicates possible violations of the given sub-task specification in the near future, and thus the MPC module is triggered. We formulate the MPC problem as follows:

**Problem 9.**

$$\begin{aligned} \min_{x(t),u(t)} \quad & \textstyle\sum_{\tau=t}^{\tau=t+T'} (x_r(\tau) - x(\tau))^T Q((x_r(\tau) - x(\tau)) \\ \textit{s.t.} \quad & x(\tau+1) = f(x(\tau), u(\tau),\ \tau \in [t, t+T'-1] \\ & x_r(t+T') = x(t+T') \end{aligned}$$

where $T'$ is the time-horizon for the MPC problem with $T' < T$. By solving Problem 9, the goal is to bring the robot back towards the reference trajectory within the acceptable tolerance range. An important detail to note here is that only the first-step of the newly computed optimal control (denoted as $u^*(t)$) is implemented at a time, i.e., at any given time $t$, when an MTL sub-task monitor detects a tolerance failure, we use $u^*(t)$ instead of $u_r(t)$ as the control input for next time step $t+1$. After that the system goes back to using the pre-computed reference control input $u_r(t)$ again. We then continue to evaluate the predicted trajectory deviations at each next time-step in an iterative fashion until the end of the sub-task planning horizon $T$. This method proved to be more efficient than using the newly computed optimal control i.e., $u^*(t)$ for the whole MPC time-horizon $T'$.

Combining MTL sub-task monitors with this event-triggered MPC mechanism, we propose that it is possible to accomplish the goal of safe learning i.e., self-monitoring and self-correction for a multiagent system in a complex mission.

## 5.5 Simulations and Results

We now demonstrate the capabilities and performance of our proposed approach, in conjun-ction with the composable MTL-based mission planning method for single and multiagent systems, which we described in detail in Chapter 2. We use two case studies in this chapter; the first one is a validation case study, where we use a single UAV in a surveillance mission scenario to showcase the various safety features of the proposed safe learning mechanism. The second one is a performance case study, where 16 UAVs are used for a multiagent surveillance mission in a compact space. The results demonstrate the efficacy of our approach in accomplishing safe and time-critical autonomy, for a team of robots that is engaged in a complex mission, with close to realtime performance. For both case studies, we use the same workspace as shown in Fig. 5.1. The experiments are run through YALMIP-CPLEX solver using MATLAB interface on an Intel NuC hades canyon. It is a portable computer with an Intel core i7 @ 3.7 GHz CPU, an integrated AMD Vega RX64 GPU, and 32 GBs of memory. This setup is transferable to a full-size quadrotor as a companion module for onboard computation. All dynamics throughout these experiments, are uniformly discretized at a rate of 2 Hz. All MTL sub-task monitors are encoded in MATLAB using the Flowstate module.

### 5.5.1 Case Study I: Validation

For this case study, we specify a time-critical surveillance mission for a single UAV in the workspace shown in Fig. 5.1. The MTL specification $\phi$ for quadrotor $q_1$ is given as:

$$\phi = \Diamond_{[0,15]}(A) \wedge \Diamond_{[0,25]}(B) \wedge \Diamond_{[0,35]}(C) \wedge \Diamond_{[0,45]}(D) \wedge \Diamond_{[0,60]}\Box(H) \wedge \Box\neg(O)$$

The mission can be summarized as follows: Starting from anywhere on the safe region $H$, the quadrotor $q_1$ needs to periodically visit certain areas of the map located at $A$, $B$, $C$, and $D$ respectively, in a confined environment. These locations are accessible to the UAV only through a number of window openings $E_i$. The task for the UAV is to safely visit the respective target areas within allocated time limits, and return to the safe zone $H$. Similar to Chapter 2 and 4, the use of prime region notation, for example $H'$ simply represents an altitude (w.r.t. $z$-axis) variation of the quadrotor while it is in the same 2D region $H$.

Following the steps presented in Chapter 2, we decompose this mission specification $\phi$ into $M$ sub-tasks $\phi^k$, where $k \in \{1, 2, ..., M\}$. In this particular case study, $M = 7$. Some of these sub-task specifications alongside their associated dynamical mode from the hybrid model (see Fig. 5.2), are listed below:

$$\phi^1 = \Box(H) \wedge \Diamond_{[0,5]}(H') \quad [mode : \textit{Take off}]$$

$$\phi^2 = \Diamond_{[0,10]}(A') \wedge \Box\neg(O) \quad [mode : \textit{Steer}]$$

$$\phi^3 = \Diamond_{[0,10]}(B') \wedge \Box\neg(O) \quad [mode : \textit{Steer}]$$

$$\phi^4 = \Diamond_{[0,10]}(C') \wedge \Box\neg(O) \quad [mode : \textit{Steer}]$$

$$\phi^5 = \Diamond_{[0,10]}(D') \wedge \Box\neg(O) \quad [mode : \textit{Steer}]$$

and so on.

It is important to note that for this surveillance mission to succeed, the UAV needs only to pass through the desired areas of interest once at any altitude. This makes sense because notice that the UAV stays in the *Steer* mode throughout the surveillance part of the mission.

Figure 5.5 shows the MTL sub-task monitors $M^{\phi^1}_{MTLsub}$ and $M^{\phi^2}_{MTLsub}$, for the sub-tasks $\phi^1$ and $\phi^2$ respectively. We build similar monitors for all the sub-task specifications $\phi^k$, where $k \in \{1, 2, ..., M\}$, which are then used for runtime monitoring of the respective sub-tasks during execution of the mission.

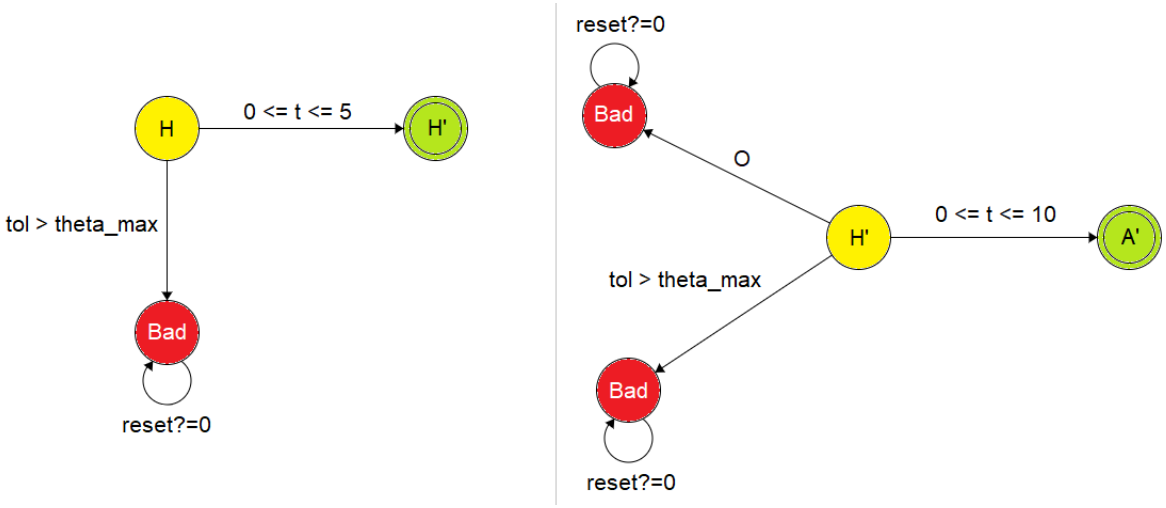


Figure 5.5: MTL sub-task monitors for the sub-tasks $\phi^1$ (left) and $\phi^2$ (right), respectively.

In the next step, we solve the resulting $M$-MILPs corresponding to each sub-task, and generate the optimal reference trajectory points $x_r(t)$ and controls $u_r(t)$ for the UAV, for all the sub-tasks in the mission (see Chapter 2 for details). We use a discrete time-horizon for the simulation as $T = 60$ units. As before, during the computation of reference trajectories, we assume perfect system and environment conditions with no presence of uncertainty. However, at the time of execution, we now introduce a white-noise deviation [102] to the system model, so as to model uncertainty during runtime. For the mission execution, we use the MPC time-horizon as $T' = 10$ units, and $\theta_{time(trig)}$ as $4$, and $\theta_{time(max)}$ as $8$ units respectively.

Figure 5.6 shows the resulting composed trajectories for the whole mission. The blue plot represents the reference trajectories for all the sub-tasks, while the red plot represents the

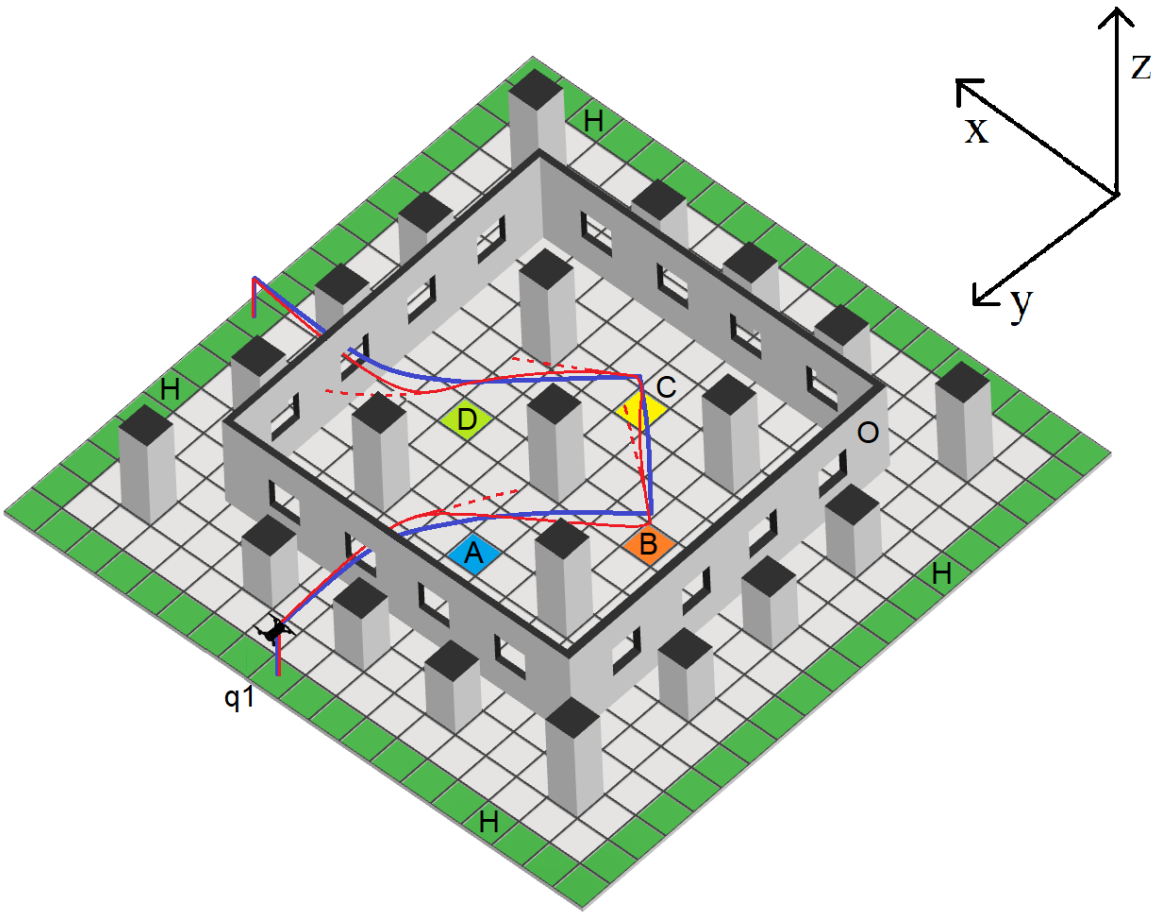


Figure 5.6: The resultant composed trajectories for the UAV-based surveillance mission with the safe learning (i.e., the self-monitoring and correction) mechanism. The blue plot represents the reference trajectory, while the red plot represents the self-corrected trajectory at runtime. The red dashed plots indicate the predicted trajectories generated at 4 different points along the way. These points represent the triggering events for the MPC module.

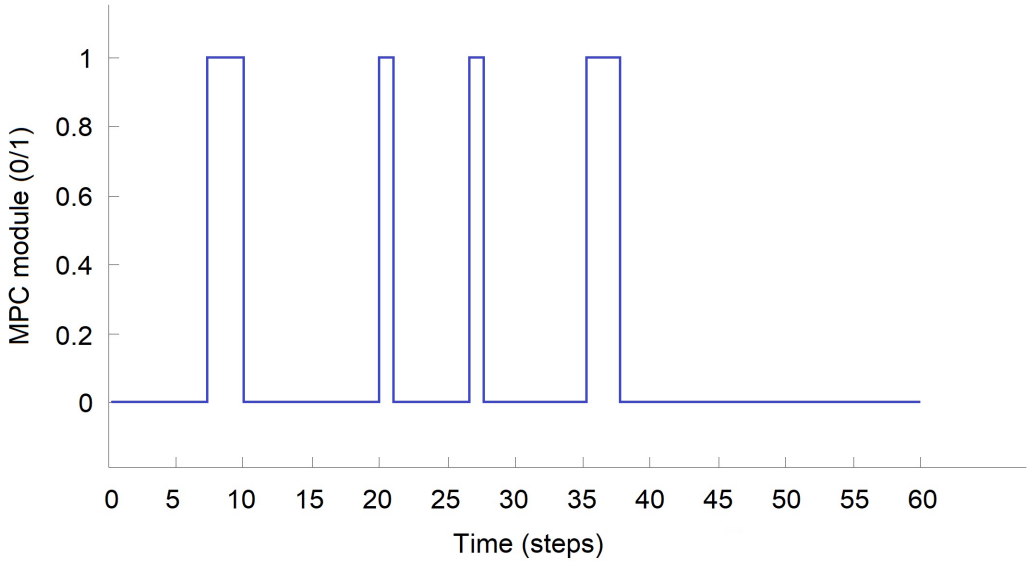


Figure 5.7: Triggering instances for the MPC module. During the complete mission the MPC module is activated 4 times, for a total duration of 8 steps.

corrected trajectories at runtime. The red dashed plots indicate the predicted trajectories at 4 different points along the way. These points represent the triggering events for the MPC module. Figure 5.7 shows the triggering instances for the MPC module over the course of the complete mission. Notice that even in the presence of added disturbance to the system, with our proposed monitoring and correction mechanism, the UAV is still able to successfully and safely visit all areas of interest while avoiding any obstacles in the environment. A closer look at the timing analysis of all the sub-tasks in Table 5.2, reveals that all timing constraints for the respective sub-tasks are also satisfied.

Notice that during the complete mission, the MPC module is turned on only 4 times, with a total duration of 8 steps. This event-triggered use of the self-correction mechanism is the key in achieving close to realtime performance. The average computation time for the optimal (reference) sub-task trajectories was recorded as approximately 4.7 seconds, while the average computation times for the predicted trajectories and for the event-triggered MPC module were

Table 5.2: Timing analysis for the sub-tasks $\phi^k$ for the whole mission

| $\mathbf{Sub-task}\ \phi^{\mathbf{k}}$ | **Execution time without safe learning (steps)** | **Execution time with safe learning (steps)** |
|---|---|---|
| $\phi^1\ (H - H')$ | $2 \leq 5$ | $2 \leq 5$ |
| $\phi^2\ (H' - A')$ | $7 \leq 10$ | $9 \leq 10$ |
| $\phi^3\ (A' - B')$ | $5 \leq 10$ | $7 \leq 10$ |
| $\phi^4\ (B' - C')$ | $5 \leq 10$ | $6 \leq 10$ |
| $\phi^5\ (C' - D')$ | $5 \leq 10$ | $6 \leq 10$ |
| $\phi^6\ (D' - H')$ | $7 \leq 10$ | $9 \leq 10$ |
| $\phi^7\ (H' - H)$ | $2 \leq 5$ | $2 \leq 5$ |

recorded as approximately 950 milliseconds and 780 milliseconds respectively. Since, all safety monitors are constructed at design-time as well, this makes the whole self-monitoing and self-correction (i.e., the safe learning) mechanism realtime implementable.

From the results of this case study, it can be observed that if the added disturbance or noise to the system at runtime is very small, the UAV can still manage to satisfy all safety and finite-time constraints within the desired tolerance levels, and self-correction is not required. However, whenever the disturbance is large enough for the system to violate the time-tolerance criteria, the MPC module is triggered, and it drives the UAV back towards the reference trajectory. Notice that due to the primary dependence of our self-correction criteria on time-tolerances, during runtime it is expected for the UAV to significantly deviate from the reference trajectory in terms of space, before the MPC module is triggered. In compact spaces with a larger number of operating agents, this can be problematic. However, this issue can be addressed to some extent by using tighter time-tolerances, which will result in more frequent triggering of the MPC module to keep the system's space-deviations in check. This idea is further portrayed in our next case study, which uses a multiagent system with the proposed safe learning mechanism in another surveillance-scenario mission.

### 5.5.2 Case Study II: Performance

We now aim to showcase the performance of our proposed approach in a multiagent setting with 16 UAVs, which are simultaneously involved in a surveillance mission in a compact space. For this case study, we do slight modifications to the mission specification and the workspace in comparison with the validation case study.

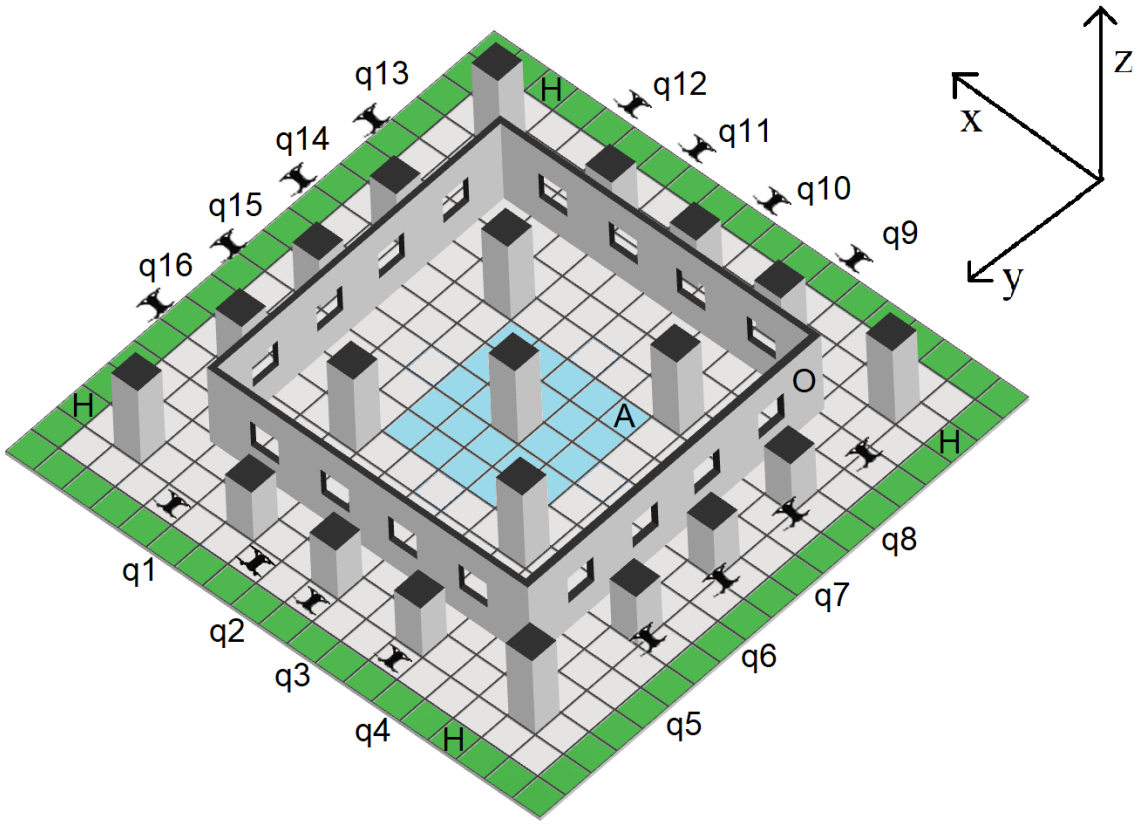


Figure 5.8: CAD model for the updated workspace used for the multiagent case study. The environment is a 19x19x3 $m^3$ workspace with a 5x5 $m^3$ area of interest $A$ at the center. 16 UAVs need to safely visit the area and return to safe zone within given, finite-time limits while operating simultaneously.

As shown in Fig. 5.8, 16 quadrotors $q_i$, where $i \in \{1, 2, ..., 16\}$, are tasked to survey an area $A$ at the center of the map, by visiting it through their respective windows $E_i$, where $i \in \{1, 2, ..., 16\}$. The UAVs start at their respective locations at the safe zone $H$, and they are

asked to visit the area of interest $A$ and return to (anywhere on) the safe zone $H$, safely and within finite-time limits. While doing so, the UAVs need to avoid any obstacles $O$, as well as the other UAVs around them. For this case study, we ignore the use of prime region notation for brevity and convenience. Therefore, for the mission to succeed, the UAVs only need to pass through the area of interest $A$ once at any altitude, as well as they can return to the safe zone $H$, at any altitude as well. For safety reasons, we limit the minimum and maximum flying altitude for all the UAVs in the *Steer* mode, to $0.5$ and $2.5m$ respectively, and the maximum flight speed to $0.5m/s$.

Given the updated workspace in Fig. 5.8, and the above description of the mission, we can write down an MTL specification $\phi_i$ for the $i^{th}$ quadrotor $q_i$ as follows:

$$\phi_i = \Diamond_{[0,20]}(A) \wedge \Diamond_{[0,50]}\Box(H) \wedge \Box\neg(O) \wedge \Box\neg q_j$$

where $i, j \in \{1, 2, ..., 16\}$, $i \neq j$, and $q_j \in \mathcal{N}_i(t)$. $\mathcal{N}_i(t)$ represents the neighborhood set of quadrotor $q_i$ i.e., all quadrotors $q_j$ s.t. $||q_i - q_j|| \leq \rho$, for some $\rho > 0$. For our experiments, we set $\rho = 0.5m$ when generating the reference trajectories.

Next, we follow once more, the same steps as discussed in the first case study, to decompose the mission specification $\phi_i$ into $M$ sub-tasks $\phi_i^k$, where $k \in \{1, 2, ..., M\}$. In this particular case study, by design, we have $M = 2$. These two sub-task specifications alongside their associated dynamical mode from the hybrid model (see Fig. 5.2), are listed below:

$$\phi_i^1 = \Diamond_{[0,20]}(A) \wedge \Box\neg(O) \wedge \Box\neg q_j \quad [mode : \textit{Steer}]$$

$$\phi_i^2 = \Diamond_{[0,30]}(H) \wedge \Box\neg(O) \wedge \Box\neg q_j \quad [mode : \textit{Steer}]$$

Figure 5.9 shows the MTL sub-task monitors $M^{\phi_i^1}_{MTLsub}$ and $M^{\phi_i^2}_{MTLsub}$, for the sub-tasks $\phi_i^1$ and $\phi_i^2$ respectively, for the $i^{th}$ quadrotor $q_i$. We build similar monitors for the sub-task specifications $\phi_i^k$, where $k \in \{1, 2\}$, for all the 16 UAVs, which are then used for runtime monitoring of the respective sub-tasks during execution of the mission.

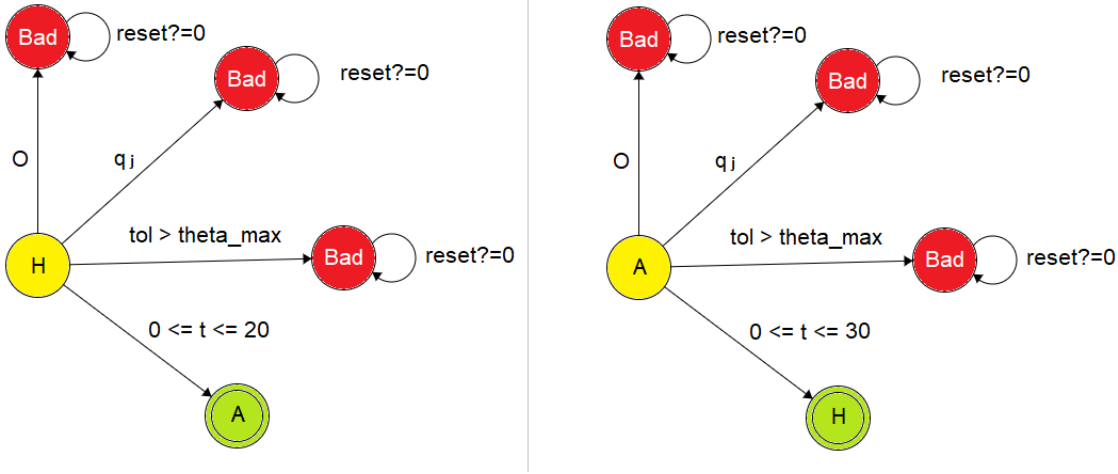


Figure 5.9: MTL sub-task monitors for the sub-tasks $\phi_i^1$ (left) and $\phi_i^2$ (right), respectively.

As was done in the validation case study, in the next step, we solve the resulting $M$-MILPs corresponding to each sub-task, and generate the optimal reference trajectory points $x_r(t)$ and controls $u_r(t)$ for all the UAVs (see Chapter 2 for details). We use a discrete time-horizon for the simulation as $T = 50$ units. As before, during the computation of reference trajectories, we assume perfect system and environment conditions with no presence of uncertainty. However, at the time of execution, we now introduce a white-noise deviation [102] to the system model of half the agents selected at random, so as to model uncertainty during runtime. For the mission execution, we use the MPC time-horizon as $T' = 10$ units, and $\theta_{time(trig)}$ as 1, and $\theta_{time(max)}$ as 4 units respectively. Notice that the time-tolerances are now selected to be much more tighter. This is due to the fact, that given the large number of agents in a close vicinity, we need to ensure

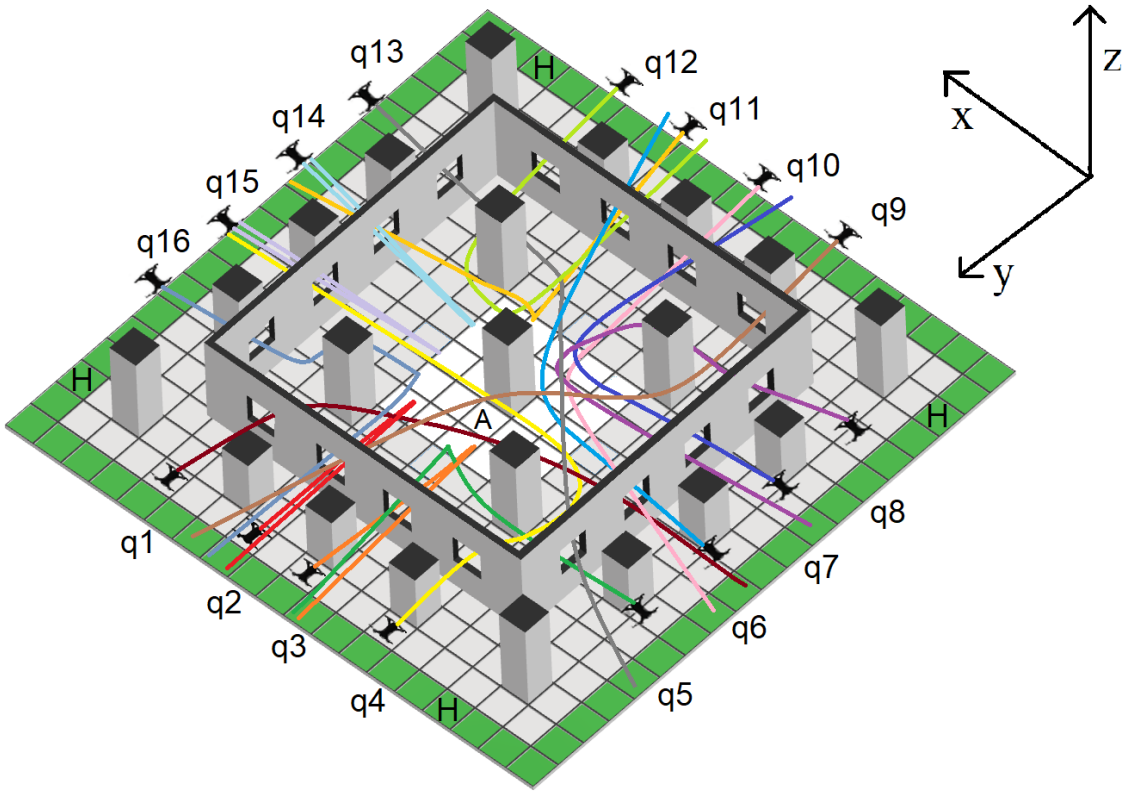


Figure 5.10: The resultant composed trajectories for the multiagent UAV-based surveillance mission with the safe learning (i.e., the self-monitoring and correction) mechanism. The various colored plots represent the collision-free, self-corrected trajectories for the UAVs at runtime.

the safety of all agents, by limiting their space-deviation from their reference trajectories, which was the case as seen earlier in validation case study. However, this trade-off comes at the expense of relatively higher number of triggering-events and respective computation times for the MPC module.

Figure 5.10 shows the resulting composed, and corrected trajectories for the 16 UAVs over the whole mission. The reference and predicted trajectories are not shown to maintain readability. For the simulation run shown in Fig. 5.10, the UAVs: $q_2$, $q_5$, $q_7$, $q_8$, $q_{10}$, $q_{11}$, $q_{12}$, and $q_{16}$ had a small white-noise disturbance added to their linear velocities. The rest of the agents did not have any external or internal disturbance. As discussed in the simulation design, due to

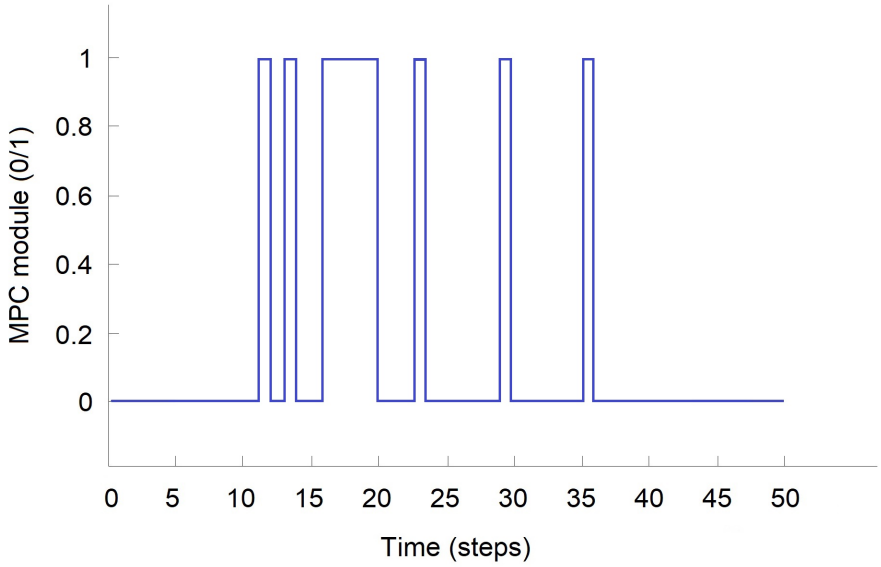


Figure 5.11: Triggering instances for the MPC module for the quadrotor $q_{11}$. During the complete mission i.e., the two sub-tasks, the MPC module is activated 6 times, for a total duration of 9 steps. This is a significant increase from the validation case study which had only 4 activations for its 7 sub-tasks in total.

the presence of tighter time-tolerances, we observe relatively high numbers of triggering events for self-correction mechanism across all agents. For example, Figure 5.11 shows the triggering instances for the MPC module over the course of the complete mission for the quadrotor $q_{11}$. Notice that even in the presence of added disturbance to the multiagent system, with our proposed monitoring and correction mechanism, the UAVs are still able to successfully and safely visit the area of interest $A$, while avoiding any obstacles and other UAVs in the environment. A closer look at the timing analysis of all the sub-tasks in Table 5.3, reveals that all timing constraints for the respective sub-tasks for all UAVs are also satisfied.

The average computation time for the optimal (reference) sub-task trajectories across all agents was recorded as approximately 12.7 seconds, while the average computation times for the predicted trajectories and for the event-triggered MPC module were recorded as approximately 2.1 and 3.5 seconds respectively. This shows a significant increase in the complexity of the

Table 5.3: Timing analysis for the sub-tasks $\phi_i^k$ for the whole mission

| **Sub − task $\phi_i^k$** | **Execution time without safe learning (steps)** | **Execution time with safe learning (steps)** |
|---|---|---|
| $\phi_1^1$ $(H-A)$ | $15 \leq 20$ | $16 \leq 20$ |
| $\phi_1^2$ $(A-H)$ | $22 \leq 30$ | $24 \leq 30$ |
| $\phi_2^1$ $(H-A)$ | $12 \leq 20$ | $12 \leq 20$ |
| $\phi_2^2$ $(A-H)$ | $12 \leq 30$ | $13 \leq 30$ |
| $\phi_3^1$ $(H-A)$ | $12 \leq 20$ | $12 \leq 20$ |
| $\phi_3^2$ $(A-H)$ | $12 \leq 30$ | $13 \leq 30$ |
| $\phi_4^1$ $(H-A)$ | $18 \leq 20$ | $19 \leq 20$ |
| $\phi_4^2$ $(A-H)$ | $24 \leq 30$ | $26 \leq 30$ |
| $\phi_5^1$ $(H-A)$ | $15 \leq 20$ | $17 \leq 20$ |
| $\phi_5^2$ $(A-H)$ | $12 \leq 30$ | $13 \leq 30$ |
| $\phi_6^1$ $(H-A)$ | $13 \leq 20$ | $14 \leq 20$ |
| $\phi_6^2$ $(A-H)$ | $24 \leq 30$ | $26 \leq 30$ |
| $\phi_7^1$ $(H-A)$ | $13 \leq 20$ | $13 \leq 20$ |
| $\phi_7^2$ $(A-H)$ | $15 \leq 30$ | $17 \leq 30$ |
| $\phi_8^1$ $(H-A)$ | $18 \leq 20$ | $18 \leq 20$ |
| $\phi_8^2$ $(A-H)$ | $22 \leq 30$ | $25 \leq 30$ |
| $\phi_9^1$ $(H-A)$ | $16 \leq 20$ | $16 \leq 20$ |
| $\phi_9^2$ $(A-H)$ | $18 \leq 30$ | $19 \leq 30$ |
| $\phi_{10}^1$ $(H-A)$ | $13 \leq 20$ | $15 \leq 20$ |
| $\phi_{10}^2$ $(A-H)$ | $20 \leq 30$ | $22 \leq 30$ |
| $\phi_{11}^1$ $(H-A)$ | $15 \leq 20$ | $16 \leq 20$ |
| $\phi_{11}^2$ $(A-H)$ | $21 \leq 30$ | $24 \leq 30$ |
| $\phi_{12}^1$ $(H-A)$ | $18 \leq 20$ | $18 \leq 20$ |
| $\phi_{12}^2$ $(A-H)$ | $23 \leq 30$ | $24 \leq 30$ |
| $\phi_{13}^1$ $(H-A)$ | $16 \leq 20$ | $19 \leq 20$ |
| $\phi_{13}^2$ $(A-H)$ | $26 \leq 30$ | $27 \leq 30$ |
| $\phi_{14}^1$ $(H-A)$ | $13 \leq 20$ | $13 \leq 20$ |
| $\phi_{14}^2$ $(A-H)$ | $13 \leq 30$ | $14 \leq 30$ |
| $\phi_{15}^1$ $(H-A)$ | $12 \leq 20$ | $12 \leq 20$ |
| $\phi_{15}^2$ $(A-H)$ | $12 \leq 30$ | $13 \leq 30$ |
| $\phi_{16}^1$ $(H-A)$ | $19 \leq 20$ | $20 \leq 20$ |
| $\phi_{16}^2$ $(A-H)$ | $15 \leq 30$ | $18 \leq 30$ |

problem as compared with the earlier case studies, which is expected in light of our earlier discussion. However, the computation and the execution performance of our method is still very competitive in its class of methods, and in all cases much better than the STL-based planning methods. The only compromise is the lack of explicit space-robustness control with MTL tasks. But this is something we knew and stated at the start of this chapter. In most cases, when the goal is to accomplish a complex and time-critical mission with a team of autonomous robots, we have shown with our approach that using MTL and its time-robustness properties is certainly enough to get the job done. As an added benefit, you can achieve close to realtime performance, which is yet to be seen with the STL-based planning methods. But if the goal is to deploy hundreds of autonomous robots at a time, such as a swarm of UAVs for instance, then our proposed approach may not be the best way to go in that case.

## 5.6 Chapter Summary

In this chapter, we have proposed an online safety and adaptation mechanism to deal with uncertainty in the environment or the system (i.e., the robot) itself, in order to maintain the spatio-temporal guarantees obtained using our MTL-based planning framework at design-time. For autonomous systems operating in dynamic environments, safety in terms of both space and time are critical requirements for assured autonomy. Therefore, in this chapter, we have proposed an online adaptation mechanism for safe execution of a complex mission by an autonomous multiagent system. We have introduced the ideas of self-monitoring and self-correction for agents using hybrid automata theory and event-triggered model predictive control (MPC). In this setting, we have proposed a composable notion of safety and adaptation for autonomous multiagent

systems, which we refer to as safe learning. We have developed a two-phase approach for our safe learning problem. In the monitoring phase, using the hybrid system model, we have built a model monitor to check whether the execution sequence at runtime matches the desired execution sequence, and also a safety monitor to check the runtime safety specifications for the system. In the correction phase, using the difference between the predicted and reference system trajectories at runtime, we have proposed an event-triggered MPC mechanism to drive the system back to the reference trajectory, so as to maintain the initial guarantees on safety and finite-time mission completion. Using two simple yet effective case studies for single and multiple agents, we have shown almost realtime performance of the proposed approach in a UAV-based surveillance and search and rescue scenario.

Based on the discussion and the results presented in this chapter, we believe that our safe learning mechanism can be used in conjunction with all three variants of the MTL-based planning framework, presented through Chapters 2-4. As an immediate direction for future work, we will continue to integrate and validate this proposed safe learning i.e., self-monitoring and self-correction mechanism with the already developed mission planning methods using MTL and STL specifications. We are also interested in the application of our approach to the problem of collaborative and aggressive flight of UAVs through a narrow passage. There, the key idea is to use the collaboration of information between two or more UAVs to help them pass through a narrow passage safely and at high speeds. For this application, we plan to use results from both Chapter 2 and Chapter 3 for the planning phase. For the runtime adaptation or learning phase, we plan to use MTL sub-task monitors for prediction. We also intend to do a comparative study between the event-triggered MPC, and the RL-based self-correction approaches.

# Chapter 6: Conclusions and Future Work

We have proposed a composable approach for planning complex missions for autonomous multiagent systems with safety and finite-time guarantees using MTL specifications. We have also proposed a safe learning or online adaptation mechanism for monitoring and correcting system behavior in a modular fashion during execution. With multiple use cases such as search and rescue, inspection tasks, cooperative navigation in a leader-follower network, and surveillance missions, we have shown the efficacy of our planning, monitoring, and correcting methods while demonstrating their close to real-time performance. Broadly speaking, we have tried to answer the question, which we set out to answer in the beginning of this thesis: "how trustworthy is the autonomy of a multi-robot system in a complex mission?" We have done so by providing formal guarantees on both safety and finite-time mission completion, for a team of autonomous UAVs and robots in almost real time.

We are currently also investigating the applications of this work to improve the safety and performance of the autonomous driving systems (ADS). Despite portraying results mostly for UAVs so far, we believe that our proposed framework is transferable to any team of autonomous agents, under some realistic assumptions. The main assumptions for this transfer are: (1) being able to represent the mission or the task as an MTL specification, and (2) having a hybrid model to represent the system dynamics. In practice, these two assumptions are very easy to satisfy for

most single and multiagent systems including the ADS. Therefore, we hope that this dissertation research will serve several modern applications of public interest, such as autonomous driving, autopilots and flight controllers, autonomous aerial grasping, and package delivery with drones etc., by improving upon the existing safety of their autonomous operation.

Throughout this thesis, we have also discussed several ideas for future directions on each of the problems that we have discussed. For example, given a finite-time constraint for the whole mission, what is the best way to divide the timing constraints among various sub-tasks. Of course it is a scheduling problem, and is dependent on many factors such as robot dynamics, its maximum attainable speed, and nature of the sub-tasks as well as the environment. For most of the work in this thesis, the individual sub-task timing constraints were constructed in a uniform fashion. An optimal way to distribute timing constraints will be a nice addition to have. Risk-aware, resilient versions of the proposed planning methods can be explored as well. Extension of this work with different tasks, dynamic obstacles other than the agents themselves, conservative time-constraints, and further studying the time-robustness properties of MTL can also yield interesting results, and are all great directions for future work.

# Bibliography

[1] Sonia Waharte and Niki Trigoni. Supporting search and rescue operations with uavs. In *2010 International Conference on Emerging Security Technologies*, pages 142–147. IEEE, 2010.

[2] Usman A Fiaz and John S Baras. A hybrid compositional approach to optimal mission planning for multi-rotor uavs using metric temporal logic. *arXiv preprint arXiv:1904.03830*, 2019.

[3] Paul E Rybski, Sascha A Stoeter, Michael D Erickson, Maria Gini, Dean F Hougen, and Nikolaos Papanikolopoulos. A team of robotic agents for surveillance. In *Proceedings of the fourth international conference on autonomous agents*, pages 9–16, 2000.

[4] Cyrill Stachniss. *Robotic mapping and exploration*, volume 55. Springer, 2009.

[5] David Lattanzi and Gregory Miller. Review of robotic infrastructure inspection systems. *Journal of Infrastructure Systems*, 23(3):04017004, 2017.

[6] Alena Otto, Niels Agatz, James Campbell, Bruce Golden, and Erwin Pesch. Optimization approaches for civil applications of unmanned aerial vehicles (uavs) or aerial drones: A survey. *Networks*, 72(4):411–458, 2018.

[7] Usman A Fiaz, Noureddine Toumi, and Jeff S Shamma. Passive aerial grasping of ferrous objects. *IFAC-PapersOnLine*, 50(1):10299–10304, 2017.

[8] Usman A Fiaz, M Abdelkader, and Jeff S Shamma. An intelligent gripper design for autonomous aerial transport with passive magnetic grasping and dual-impulsive release. In *AIM 2018*, pages 1027–1032. IEEE, 2018.

[9] Usman A Fiaz and Jeff S Shamma. The usbot and programmable self-assembly. In *2018 IEEE International Conference on Robotics and Automation (ICRA) Workshop on Swarms: From Biology to Robotics and Back. IEEE*, 2018.

[10] Usman A Fiaz and Jeff S Shamma. Usbot: A modular robotic testbed for programmable self-assembly. *IFAC-PapersOnLine*, 52(15):121–126, 2019.

[11] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[12] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[13] Purushothaman Raja and Sivagurunathan Pugazhenthi. Optimal path planning of mobile robots: A review. *International journal of physical sciences*, 7(9):1314–1320, 2012.

[14] Dustin J Webb and Jur van den Berg. Kinodynamic rrt*: Asymptotically optimal motion

planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5054–5061. IEEE, 2013.

[15] Yong Koo Hwang, Narendra Ahuja, et al. A potential field approach to path planning. *IEEE transactions on robotics and automation*, 8(1):23–32, 1992.

[16] Shuzhi Sam Ge and Yan Juan Cui. New potential functions for mobile robot path planning. *IEEE Transactions on robotics and automation*, 16(5):615–620, 2000.

[17] Wei Xi, Xiaobo Tan, and John S Baras. A hybrid scheme for distributed control of autonomous swarms. In *ACC 2005*, pages 3486–3491. IEEE, 2005.

[18] Jur van den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[19] Hai Zhu and Javier Alonso-Mora. Chance-constrained collision avoidance for mavs in dynamic environments. *IEEE Robotics and Automation Letters*, 4(2):776–783, 2019.

[20] Wolfgang Hönig, James A Preiss, TK Satish Kumar, Gaurav S Sukhatme, and Nora Ayanian. Trajectory planning for quadrotor swarms. *IEEE Transactions on Robotics*, 34(4):856–869, 2018.

[21] Senthil Hariharan Arul and Dinesh Manocha. Swarmcco: Probabilistic reactive collision avoidance for quadrotor swarms under uncertainty. *IEEE Robotics and Automation Letters*, 6(2):2437–2444, 2021.

[22] Jesus Tordesillas and Jonathan P How. Mader: Trajectory planner in multiagent and dynamic environments. *IEEE Transactions on Robotics*, 38(1):463–476, 2021.

[23] Jesus Tordesillas, Brett T Lopez, and Jonathan P How. Faster: Fast and safe trajectory planner for flights in unknown environments. In *2019 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 1934–1940. IEEE, 2019.

[24] Konstantinos Kanistras, Goncalo Martins, Matthew J Rutherford, and Kimon P Valavanis. A survey of unmanned aerial vehicles (uavs) for traffic monitoring. In *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 221–234. IEEE, 2013.

[25] Samira Hayat, Evşen Yanmaz, Timothy X Brown, and Christian Bettstetter. Multi-objective uav path planning for search and rescue. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 5569–5574. IEEE, 2017.

[26] Christel Baier, Joost-Pieter Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[27] Chad Goerzen, Zhaodan Kong, and Bernard Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *JIRS*, 57(1-4):65, 2010.

[28] Yiannis Kantaros and Michael M Zavlanos. Temporal logic optimal control for large-scale multi-robot systems: 10 400 states and beyond. In *CDC*, pages 2519–2524. IEEE, 2018.

[29] Hadas Kress-Gazit, Georgios E Fainekos, and George J Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE transactions on robotics*, 25(6):1370–1381, 2009.

[30] Yushan Chen, Xu Chu Ding, Alin Stefanescu, and Calin Belta. Formal approach to the deployment of distributed robotic teams. *IEEE Transactions on Robotics*, 28(1):158–171, 2011.

[31] Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 854–860. Association for Computing Machinery, 2013.

[32] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

[33] Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In *Formal Modeling and Analysis of Timed Systems*, pages 1–13. Springer, 2008.

[34] Sertac Karaman, Ricardo G Sanfelice, and Emilio Frazzoli. Optimal control of mixed logical dynamical systems with ltl specifications. In *CDC*, pages 2117–2122. IEEE, 2008.

[35] Eric M Wolff, Ufuk Topcu, and Richard M Murray. Optimization-based trajectory generation with ltl specifications. In *ICRA 2014*, 2014.

[36] Dipankar Maity and John S Baras. Motion planning in dynamic environments with bounded time temporal logic specifications. In *MED 2015*, pages 940–946. IEEE, 2015.

[37] Yuchen Zhou, Dipankar Maity, and John S Baras. Optimal mission planner with timed temporal logic constraints. In *2015 European Control Conference (ECC)*. IEEE, 2015.

[38] Alexandros Nikou, Jana Tumova, and Dimos V Dimarogonas. Cooperative task planning of multi-agent systems under timed temporal specifications. In *ACC*. IEEE, 2016.

[39] Lars Lindemann, Dipankar Maity, John S Baras, and Dimos V Dimarogonas. Event-

triggered feedback control for signal temporal logic tasks. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 146–151. IEEE, 2018.

[40] Lars Lindemann, George J Pappas, and Dimos V Dimarogonas. Reactive and risk-aware control for signal temporal logic. *IEEE Transactions on Automatic Control*, 2021.

[41] Rajeev Alur. *Principles of cyber-physical systems*. MIT Press, 2015.

[42] Nathan Michael, Daniel Mellinger, Quentin Lindsey, and Vijay Kumar. The grasp multiple micro-uav testbed. *IEEE Robotics & Automation Magazine*, 17(3):56–65, 2010.

[43] Luis Rodolfo García, Alejandro Enrique Dzul López, Rogelio Lozano, and Claude Pégard. Modeling the quad-rotor mini-rotorcraft. *Quad Rotorcraft Control*, pages 23–34, 2013.

[44] Usman A Fiaz and John S Baras. Fast, composable rescue mission planning for uavs using metric temporal logic. *IFAC-PapersOnLine*, 53(2):15404–15411, 2020.

[45] Usman Amin Fiaz and Jeff S Shamma. Servo-actuated rotary magnetic latching mechanism and method, June 10 2021. US Patent App. 16/761,634.

[46] Usman Amin Fiaz, Jeff S Shamma, and Mohamed A Abdelkader. Impulsive release mechanism and method, February 1 2022. US Patent 11,235,873.

[47] Usman A. Fiaz. Passive magnetic latching mechanisms for robotic applications. Master's thesis, KAUST, 2017.

[48] Jana Tumova and Dimos V Dimarogonas. Decomposition of multi-agent planning under distributed motion and task ltl specifications. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 7448–7453. IEEE, 2015.

[49] Philipp Schillinger, Mathias Bürger, and Dimos V Dimarogonas. Decomposition of finite ltl specifications for efficient multi-agent planning. In *DARS*, pages 253–267. Springer, 2018.

[50] Gerd Behrmann, A. David, P. Pettersson, W. Yi, and M. Hendriks. Uppaal 4.0. 2006.

[51] Vijay Kumar, Daniela Rus, and Sanjiv Singh. Robot and sensor networks for first responders. *IEEE Pervasive computing*, 3(4):24–33, 2004.

[52] Daniel P Stormont. Autonomous rescue robot swarms for first responders. In *CIHSPS 2005. Proceedings of the 2005 IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety, 2005.*, pages 151–157. IEEE, 2005.

[53] Christian Willms, Constantin Houy, Jana-Rebecca Rehse, Peter Fettke, and Ivana Kruijff-Korbayová. Team communication processing and process analytics for supporting robot-assisted emergency response. In *2019 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 216–221. IEEE, 2019.

[54] Daniel Pickem, Paul Glotfelter, Li Wang, Mark Mote, Aaron Ames, Eric Feron, and Magnus Egerstedt. The robotarium: A remotely accessible swarm robotics research testbed. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1699–1706. IEEE, 2017.

[55] Mehran Mesbahi and Magnus Egerstedt. *Graph theoretic methods in multiagent networks*, volume 33. Princeton University Press, 2010.

[56] Daniel S Drew. Multi-agent systems for search and rescue applications. *Current Robotics Reports*, 2(2):189–200, 2021.

[57] Angela Davids. Urban search and rescue robots: from tragedy to technology. *IEEE Intelligent systems*, 17(2):81–83, 2002.

[58] James S Jennings, Greg Whelan, and William F Evans. Cooperative search and rescue with a team of mobile robots. In *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97*, pages 193–200. IEEE, 1997.

[59] George Kantor, Sanjiv Singh, Ronald Peterson, Daniela Rus, Aveek Das, Vijay Kumar, Guilherme Pereira, and John Spletzer. Distributed search and rescue with robot and sensor teams. In *Field and Service Robotics*, pages 529–538. Springer, 2003.

[60] Rodrigo Ventura and Pedro U Lima. Search and rescue robots: The civil protection teams of the future. In *2012 Third International Conference on Emerging Security Technologies*, pages 12–19. IEEE, 2012.

[61] Bingxi Li, Sharvil Patankar, Barzin Moridian, and Nina Mahmoudian. Planning large-scale search and rescue using team of uavs and charging stations. In *2018 IEEE international symposium on safety, security, and rescue robotics (SSRR)*, pages 1–8. IEEE, 2018.

[62] Zendai Kashino, Goldie Nejat, and Beno Benhabib. Aerial wilderness search and rescue with ground support. *Journal of Intelligent & Robotic Systems*, 99(1):147–163, 2020.

[63] Robert Bogue. Disaster relief, and search and rescue robots: the way forward. *Industrial Robot: the international journal of robotics research and application*, 2019.

[64] Jennifer L Casper, Mark Micire, and Robin R Murphy. Issues in intelligent robots for search and rescue. In *Unmanned ground vehicle technology II*, volume 4024, pages 292–302. SPIE, 2000.

[65] Francesco Bullo, Jorge Cortés, and Sonia Martinez. *Distributed control of robotic networks: a mathematical approach to motion coordination algorithms*, volume 27. Princeton University Press, 2009.

[66] Dimos V Dimarogonas and Kostas J Kyriakopoulos. On the rendezvous problem for multiple nonholonomic agents. *IEEE Transactions on automatic control*, 52(5):916–922, 2007.

[67] Tove Gustavi, Dimos V Dimarogonas, Magnus Egerstedt, and Xiaoming Hu. Sufficient conditions for connectivity maintenance and rendezvous in leader–follower networks. *Automatica*, 46(1):133–139, 2010.

[68] Zhiyun Lin, Wei Ding, Gangfeng Yan, Changbin Yu, and Alessandro Giua. Leader–follower formation via complex laplacian. *Automatica*, 49(6):1900–1906, 2013.

[69] Sung G Lee, Yancy Diaz-Mercado, and Magnus Egerstedt. Multirobot control using time-varying density functions. *IEEE Transactions on Robotics*, 31(2):489–493, 2015.

[70] Xiaotian Xu and Yancy Diaz-Mercado. Multi-agent control using coverage over time-varying domains. In *2020 American Control Conference (ACC)*, pages 2030–2035. IEEE, 2020.

[71] Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil

Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European control conference (ECC)*, pages 3420–3431. IEEE, 2019.

[72] Der-Tsai Lee and Bruce J Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.

[73] Anousheh Gholami, Usman A Fiaz, and John S Baras. Drone-assisted communications for remote areas and disaster relief. *arXiv preprint arXiv:1909.02150*, 2019.

[74] Mohamed Abdelkader, Usman A Fiaz, Noureddine Toumi, Mohamed A Mabrok, and Jeff S Shamma. Riscuer: a reliable multi-uav search and rescue testbed. In *Unmanned Aerial Systems*, pages 345–374. Elsevier, 2021.

[75] Guillaume Hoareau, Johannes J Liebenberg, John G Musial, and Todd R Whitman. Package transport by unmanned aerial vehicles, August 15 2017. US Patent 9,731,821.

[76] Mohammad Mozaffari, Walid Saad, Mehdi Bennis, and Mérouane Debbah. Efficient deployment of multiple unmanned aerial vehicles for optimal wireless coverage. *IEEE Communications Letters*, 20(8):1647–1650, 2016.

[77] Jingjing Zhang, Liang Liu, Binhai Wang, Xiguang Chen, Qian Wang, and Tianru Zheng. High speed automatic power line detection and tracking for a uav-based inspection. In *2012 International Conference on Industrial Control and Electronics Engineering*, pages 266–269. IEEE, 2012.

[78] G Morgenthal and N Hallermann. Quality assessment of unmanned aerial vehicle (uav)

based visual inspection of structures. *Advances in Structural Engineering*, 17(3):289–302, 2014.

[79] Xiaoxia Li, Qiang Yang, Zhebo Chen, Xuejing Luo, and Wenjun Yan. Visible defects detection based on uav-based inspection in large-scale photovoltaic systems. *IET Renewable Power Generation*, 11(10):1234–1244, 2017.

[80] Yunpeng Wu, Yong Qin, Zhipeng Wang, and Limin Jia. A uav-based visual inspection method for rail surface defects. *Applied sciences*, 8(7):1028, 2018.

[81] Timo Rolf Bretschneider and Karan Shetti. Uav-based gas pipeline leak detection. In *Proc. of ARCS*, 2015.

[82] Brodie Chan, Hong Guan, Jun Jo, and Michael Blumenstein. Towards uav-based bridge inspection systems: A review and an application perspective. *Structural Monitoring and Maintenance*, 2(3):283–300, 2015.

[83] Sainab Feroz and Saleh Abu Dabous. Uav-based remote sensing applications for bridge condition assessment. *Remote Sensing*, 13(9):1809, 2021.

[84] Huai Yu, Wen Yang, Heng Zhang, and Wanjun He. A uav-based crack inspection system for concrete bridge monitoring. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 3305–3308. IEEE, 2017.

[85] J.C. Latombe. *Robot motion planning*. Kluwer international series in engineering and computer science. Kluwer Academic Publishers, Boston, 1991.

[86] Marko Lepetič, Gregor Klančar, Igor Škrjanc, Drago Matko, and Boštjan Potočnik. Time

optimal path planning considering acceleration limits. *Robotics and Autonomous Systems*, 45(3-4):199–210, 2003.

[87] Erion Plaku and Sertac Karaman. Motion planning with temporal-logic specifications: Progress and challenges. *AI communications*, 29(1):151–162, 2016.

[88] Sayan Saha and A Agung Julius. An milp approach for real-time optimal controller synthesis with metric temporal logic specifications. In *2016 American Control Conference (ACC)*, pages 1105–1110. IEEE, 2016.

[89] Sofie Andersson, Alexandros Nikou, and Dimos V Dimarogonas. Control synthesis for multi-agent systems under metric interval temporal logic specifications. *IFAC-PapersOnLine*, 50(1):2397–2402, 2017.

[90] Keliang He, Andrew M. Wells, Lydia E. Kavraki, and Moshe Y. Vardi. Efficient symbolic reactive synthesis for finite-horizon tasks. In *IEEE Intl. Conf. on Robotics and Automation*, pages 8993–8999, May 2019. (Best paper award in Cognitive Robotics).

[91] Linda G Shapiro, George C Stockman, et al. *Computer vision*, volume 3. Prentice Hall New Jersey, 2001.

[92] Duane Q. Nykamp. The arc length of a parametrized curve. `https://mathinsight.org/parametrized_curve_arc_length`.

[93] Usman A Fiaz and John S Baras. Assured autonomy and safe mission planning for multiagent systems. In *2021 Maryland Robotics Center Research Symposium*, 2021.

[94] Stefan Mitsch, Khalil Ghorbal, and André Platzer. On provably safe obstacle avoidance

for autonomous robotic ground vehicles. In *Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24-June 28, 2013*, 2013.

[95] Stefan Mitsch and André Platzer. Modelplex: Verified runtime validation of verified cyber-physical system models. *Formal Methods in System Design*, 49(1):33–74, 2016.

[96] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(4):1–64, 2011.

[97] Zhenyu Lin and John S Baras. Optimization-based motion planning and runtime monitoring for robotic agent with space and time tolerances. *IFAC-PapersOnLine*, 53(2):1874–1879, 2020.

[98] Zhenyu Lin and John S Baras. Metric interval temporal logic based reinforcement learning with runtime monitoring and self-correction. In *2020 American Control Conference (ACC)*, pages 5400–5406. IEEE, 2020.

[99] Li Li, Kangye Qu, and Kuo-Yi Lin. A survey on attack resilient of uav motion planning. In *2020 IEEE 16th International Conference on Control & Automation (ICCA)*, pages 558–563. IEEE, 2020.

[100] Lifeng Zhou and Pratap Tokekar. Risk-aware submodular optimization for multirobot coordination. *IEEE Transactions on Robotics*, 2022.

[101] Stefano Primatesta, Giorgio Guglieri, and Alessandro Rizzo. A risk-aware path planning strategy for uavs in urban environments. *Journal of Intelligent & Robotic Systems*, 95(2):629–643, 2019.

[102] Zhenyu Lin. *Planning, Monitoring and Learning with Safety and Temporal Constraints for Robotic Systems*. PhD thesis, 2019.

[103] Andreas Bauer, Martin Leucker, and Christian Schallhart. Comparing ltl semantics for runtime verification. *Journal of Logic and Computation*, 20(3):651–674, 2010.

[104] Alëna Rodionova, Lars Lindemann, Manfred Morari, and George J Pappas. Time-robust control for stl specifications. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 572–579. IEEE, 2021.

[105] Alëna Rodionova, Lars Lindemann, Manfred Morari, and George J Pappas. Temporal robustness of temporal logic specifications: Analysis and control design. *arXiv preprint arXiv:2203.15661*, 2022.