SPECIFICATION-BASED TASK ORCHESTRATION FOR MULTI-ROBOT AERIAL TEAMS

A Dissertation
Presented to
The Academic Faculty

By

Christopher Banks

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology

December 2022

SPECIFICATION-BASED TASK ORCHESTRATION FOR MULTI-ROBOT AERIAL TEAMS

Thesis committee:

Dr. Magnus Egerstedt
Electrical and Computer Engineering
University of California, Irvine

Dr. Ye Zhao
Mechanical Engineering
Georgia Institute of Technology

Dr. Samuel Coogan
Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Seth Huchinson
Interactive Computing
Georgia Institute of Technology

Dr. Sonia Chernova
Interactive Computing
Georgia Institute of Technology

Date approved: August 4, 2022

”No man ever steps in the same river twice, for it’s not the same river and he’s not the same man.” - Heraclitus.

To my family: Mom, Dad, and Joshua

ACKNOWLEDGMENTS

When I reflect on my PhD journey, the famous proverb "It takes a village to raise a child" comes to mind regarding my progress as a researcher and the work within this thesis. This dissertation would not be possible without the support of friends, friends of my family, family, and colleagues. There are not enough words to express my gratitude towards you all.

First, I am honored to have worked with my primary advisor, Dr. Magnus Egerstedt. His enthusiasm and philosophy for teaching and research were instrumental to my growth as a PhD student. He is a great mentor and researcher and provided an excellent insight into how to run a world class research lab with motivated students. I will always cherish the insights and advice he shared with me over the past five years. I'd like to thank my co-advisor, Dr. Samuel Coogan for accepting me into the FACTS Lab and being an amazing mentor who focused on the details of research and pushed me to explore the boundaries of my experience and knowledge. I am grateful for the opportunity to work with him and his dedication to seeing my success as a researcher.

I would like to acknowledge the thesis committee: Dr. Sonia Chernova, Dr. Seth Hutchinson and Dr. Ye Zhao. Thank you for your feedback through both my proposal and throughout my research endeavors. Your support helped to improve the quality of my thesis and provided valuable assistance in my progress as a researcher.

My progress through the PhD would not have been nearly as fulfilling without the current and former lab members of the GRITS and FACTS Labs. You all will always be important to me and have a special place in my heart and mind, particularly, the following people:

- Ian Buckley who was the very first person I met from the GRITS Lab and immediately welcomed me into the lab with open arms. I will never forget your awesome sense of humor and our shared love of aquariums.

- Sebastian Ruf for being the TA for my controls courses at GT and being a great resource for me as an incoming PhD student.
- Li Wang, who introduced me to quadrotor research.
- Kyle Slovak who took controls courses with me and actively worked with me on my first research paper at Georgia Tech.
- Paul Glotfelter who helped me during my qualifying exams and shared many laughs as lab mates and desk mates.
- Maria Santos who showed me the importance of presentation style and who was a patient teacher while I practiced Spanish with her.
- Gennaro Notomista for his amazing math jokes, brilliant mind and infectious enthusiasm. I'll always remember our laughs and late nights working in the lab.
- Siddharth Mayya for his meticulous nature and sharp wit. I'll never forget our escapade to Hong Kong and the exciting time we had there.
- Sean Wilson for being an amazing mentor and the glue that holds the Robotarium team together.
- Yousef Emam for sharing the joys and trials we both experienced during our time in the PhD.
- Pietro Pierpaoli for his amazing cooking skills and unwavering enthusiasm to progress his research.
- Mohit Srinivasan for his tire stories.
- Cesar Santoyo who entered the PhD during the same time as me and shared many ups and downs during our time here. I'll always remember our enlightening chats about Warren Buffett and investments.

- Matthew Abate for his sharp mind for math and openness with feedback for Robo-Grads.
- Carmen Jiminez for working with me on the first FACTS Lab retreat.
- Rohit Konda who told many good jokes and is a surprisingly good paintball player.

I'd like to thank the Georgia Tech community and the Robotics PhD Program for the support systems they provided and the amazing friends and colleagues I have made during my time here. I'd like to acknowledge the following friends: Mouhyemen Khan, Ambuz Vimal, Aditya Kambil, Akash Patel, Moamen Soliman, Nicolas Shu, Nathan Glaser, Luis Rosa, William Sealy and Bogdan Vlahov. You all have been so instrumental in my growth as a person and as an academic scholar, I could not have met better friends while here at Georgia Tech. From late nights studying control theory to arguments over the best anime, my time here has truly been enriched by knowing you all. I'd like to thank my close friends Sasha Moore, Devon Denner and Brendon Johnson for continually supporting me and making my visits back home to Virginia memorable. Also, I'd like to thank D.C., my turtle, for always being the star of the show no matter the occasion.

Lastly, I would like to thank my family: my Mom – whose altruism goes beyond definition, my Dad – whose stoicism speaks volumes, and Joshua, my brother, for their continual support and encouragement throughout my PhD journey. Thank you for being there for me and having unwavering faith in my abilities to pursue my passions. Mom and Dad, you both are amazing role models and have taught me so much about the world and about life. I love you all, thank you!

"...the race is not to the swift, nor the battle to the strong...but time and chance happeneth to them all. For man also knoweth not his time..." - Ecclesiastes 9:11-12 (KJV)

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

As humans begin working more frequently in environments with multi-agent systems, they are presented with challenges on how to control these systems in an intuitive manner. Current approaches tend to limit either the interaction ability of the user or limit the expressive capacity of instructions given to the robots. Applications that utilize temporal logics provide a human-readable syntax for systems that ensures formal guarantees for specification completion. By providing a modality for global task specification, we seek to reduce cognitive load and allow for high-level objectives to be communicated to a multi-agent system. In addition to this, we also seek to expand the capabilities of swarms to understand desired actions via interpretable commands retrieved from a human.

In this thesis, we first present a method for specification-based control of a quadrotor. We utilize quadrotors as a highly agile and maneuverable application platform that has a wide variety of uses in complex problem domains. Leveraging specification-based control allows us to formulate a specification-based planning framework that will be utilized throughout the thesis. We then present methods for creating systems which allows us to provide task decomposition, allocation and planning for a team of quadrotors defined as *task orchestration* of multi-robot systems. Next, the task allocation portion of the task orchestration work is extended in the online case by considering cost agnostic sampling of trajectories from an online optimization problem. Then, we will introduce learning techniques where temporal logic specifications are learned and generated from a set of user given traces. Finally, we will conclude this thesis by presenting an extension to the Robotarium through hardware and software modifications that provides remote users access to control aerial swarms.

CHAPTER 1
INTRODUCTION

In human swarm interaction, users assign control laws or actions to single or multiple agents in a swarm. Ideal interactions between a human and swarm of robots reduce the cognitive load on the user and provide a interface for commands to be sent to robots in an efficient and scalable manner [1]. One of the most important concepts in human swarm interaction is interaction modality type. Indeed, in [2, 3] studies are conducted on modality types and user preferred interactions. We show in this thesis that our formulation of specification-based planning, in the design and application of high-level specifications to robotic systems, is amenable to multi-robot control for complex robotics systems (e.g. quadrotors). In this thesis we explore the idea of specification-based planning and control of quadrotors which involves generating trajectories under system [4, 5, 6] and environmental constraints [7, 8] for high-level goals generated as task specifications, task allocation and decomposition in the multi-agent case and leveraging alternative methods of specification design [9, 10] such as learning techniques to make specifications informed from data.

Well defined control modalities are essential for human interaction with multi-robot systems. In [2], the authors compare two categories of human swarm interaction: selection-based control and beacon based control. Selection-based control allows users to select a leader or subset of leaders from a swarm and influence the swarm behavior through a leader follower framework [11, 12, 13]. Beacon-based control relies on influencing the global behavior of swarm through the introduction of forces on the swarm through indirect control. In the literature, we can see many examples where other types of human swarm interactions are developed such as gesture control of robotic swarms [14, 15] or parameterized learning of global swarm behaviors [16]. This thesis focuses on the use of specification-based planning [17, 18, 19] where users provide high level global goals to a system and planning

and allocation are demonstrated at the system level. This type of interaction modality lends itself to a reduction in frequency of swarm interaction from the human perspective, development of desired generalized behaviors defined by the specification designer and ease of deployment once generated. For mobile robotic systems this helps maintain system performance as desired behaviors can easily be encapsulated as specifications. For quadrotors, these interactions are even more critical to desired system performance from the users perspective. Ideal interactions must consider the dynamical constraints of each quadrotor to ensure safe [20, 21] and executable commands. Therefore, in the first part of this thesis we will develop a methodology for designing trajectories defined for high-level task specifications. By informing trajectory design with a task specification and system constraints, for certain specifications, we can guarantee system performance and task satisfaction. Our choice of specification in this paper are linear temporal logic formulas, a logic formalism well suited for specifying desired linear-time properties of a system [22, 23].

Temporal logic control of swarms has been approached in a variety of ways by many researchers. In these approaches, researchers leverage the well defined safety and liveliness properties of temporal logic systems [24, 22, 25] and apply them to robotic systems. These approaches have been applied to motion planning both in the discrete [26] and continuous [27] space where systems must satisfy Boolean propositions defined over the problem domain. For well defined environments and systems, generated as transition systems from a set of propositions, temporal logic specifications can be used to find sequences of states that satisfy these transition systems as well as the goal specification defined as a temporal logic specification [28, 29]. Therefore, in Chapter 3 we will use temporal logic as a task specification language for a single quadrotor for reconfigurable waypoints in three-dimensional space. Then, we will extend our results to the multi-agent case and consider how to utilize task decomposition and allocation using temporal logics. Prior works that utilize task decomposition using temporal logic focus on distributed algorithm design [30, 31], individual agent specifications [32] or sequential satisfaction of specifications over a pre-defined time

period [33]. In Chapter 4 we will utilize the global goal specifications defined for the entire system and decompose tasks based on optimal agent assignments based on cost functions assigned to each agent a-priori. Then, from optimal agent assignments we consider the case of unknown optimal costs for each agent. These situations may arise from use by novice users where system behavior may not be fully understood or difficult to characterize via an optimal cost without consulting an expert [34]. We develop online cross entropy task allocation for multi-agent systems in Chapter 5 where optimal costs are learned on-the-fly for each agent in a multi-agent systems and we find optimal task assignments for a team of quadrotors.

Building on the idea that users may be unfamiliar with system behavior, we also consider cases when users may have desired goals for a robotic system but are unable to design a desired specification. Temporal logic specifications can be difficult to design for the novice user and often require expert knowledge for generating specifications that capture desired system performance. There is a broad array of research, in the computer science literature, on temporal logic mining [35, 36, 37]. In this context, researchers seek to find specifications that satisfy a set of user provided trace logs. In robotics literature this problem has been explored as temporal logic learning [10, 38, 39] where models and feature spaces are defined to learn temporal logic specifications as opposed to the brute force methods commonly seen in temporal logic mining literature. In Chapter 6 we approach the temporal logic learning problem using support vector machines (SVMs) as a classification method where trace features are used to map traces to temporal logic specifications. Via this approach, we provide a method for novice users to define desired system behavior with little knowledge of temporal logic and only provide desired trace information instead. In Chapter 7 we consider how to provide external users access to aerial vehicles on the Robotarium while guaranteeing safety. By developing waypoint control and providing a minimally invasive control modality, novice users are allowed to control aerial teams on the Robotarium.

1.1 Thesis Statement

This dissertation presents methods for specification-based control of single and multi-robot aerial vehicles that are cost-aware and leverage highly expressive task assignments provided by end users.

1.2 Contributions

The contributions of this thesis are the following:

- **Specification-based maneuvering for single quadrotors** (Chapter 3): We formally develop a specification-based control technique that leverages the dynamical constraints of a quadrotor in addition to user provided constraints through cost functions applied over desired trajectories. This technique leverages temporal logic for reconfigurable waypoints as a method of providing highly expressive task assignment to a quadrotor. This technique was verified through experiments on physical quadrotors and objective locations.

- **Development of a multi-agent task orchestration framework** (Chapter 4): We develop a task orchestration framework which is defined as task decomposition, allocation and planning for a multi-agent system. Tasks are allocated based on the stochastic optimization technique, cross-entropy optimization. The framework is verified through a case study on fire-fighting quadrotor teams.

- **Efficient improvement on multi-agent task allocation** (Chapter 5): The allocation method for quadrotors is improved via an online methodology where agents are considered at each time step. The results of this method show that the online method is faster and presents less expensive solutions to the task allocation problem than the offline solution.

- **Evaluation of a specification learning model** (Chapter 6): A specification learning

model is presented that leverages SVMs for learning specifications given user traces. This model is evaluated against brute force techniques where it is shown to be more consistent in performance over variable specification sizes and faster compared to techniques implemented in the same language. In addition to this, we show that testing accuracy for this model is 91% for traces labeled with the correct template type. The model performance is also evaluated via a house surveillance case study.

- **Development of a remotely accessible aerial swarm testbed** (Chapter 7): We develop an extension to the Robotarium by providing remote access to aerial swarms. A fully autonomous system is developed that provides users safe, minimally invasive access to multiple quadrotors. The results of this system are shown via the types of networked robotics experiments capable of being run on the Robotarium in addition to the access provided via the simulators.

1.3 Outline of Dissertation Document

The thesis is organized as follows. In Chapter 3, we introduce *specification-based maneuvering* for quadrotors where we leverage high level specifications to generate plans for quadrotors on reconfigurable waypoints. We follow this work with *task orchestration*, defined in Chapter 4 where we consider simultaneous task decomposition, allocation and planning for a team of quadrotors given high level specifications. We then extend this work in Chapter 5 by introducing online task allocation via cross entropy optimization. In introducing online cross entropy optimization to this framework we consider task allocation for optimal assignments without costs defined a-priori by end users. We introduce a form of LTL learning in Chapter 6 where we provide a method of learning high-level temporal logic specifications through user provided traces. Finally, in Chapter 7 we present hardware and software modifications made on the Robotarium to provide external users remote access to aerial swarms. Chapter 8 concludes this thesis by outlining the main contributions and future work.

CHAPTER 2
BACKGROUND

In this chapter we will review the relevant literature pertaining to human swarm interaction and the use of temporal logic specification-based planning for robotic systems. We will first introduce human swarm interaction, ending on specification-based goals provided by the end user. Then, we will introduce path planning and control for quadrotors. We will address other works that have utilized temporal logic for providing high level specifications to single agents and robot teams. Finally, we will end this chapter with temporal logic specification learning where learning models are provided with data about a system (e.g. desired traces from successful system execution) and must learn what specifications will satisfy this input data.

2.1 Human Swarm Interaction

As large groups of mobile robots are more frequently deployed out in the field, humans or groups of humans need to find ideal interactions with these systems. Human swarm interaction is a field of robotics focusing on developing interaction modalities between humans and robotics systems [1]. Swarms can be applied to solve a myriad of real world problems such as search and rescue [40, 41], reconnaissance [8, 42], precision agriculture [43] and even warehouse operations [44]. By leveraging the emergent properties of swarms – unexpected behaviors that arise due to the complex interactions between agents and their environment – researchers in human swarm interaction avoid encoding explicit behavior in their systems. In addition to this, swarms also provide robustness to single agent failure or disturbances in the environment, faster execution time for many problem domains and provide redundancies in task allocation.

There exists a wide variety of interaction modalities for human control of swarms. Sur-

vey papers on the topic [1, 45] tend to generalize interactions to the following: behavior selection, leader selection, environmental control and parameter setting. In behavior selection, individual agents or a subset of agents are selected and given particular algorithms or commands separate from the group behavior. This is seen in works like [14] where a gesture recognition framework is developed for multi-UAV interaction and individual agents are selected to execute designated behaviors. In [3] various types of control modalities for collaborative interaction with UAVs (unmanned aerial vehicles) were studied to determine the level of information necessary to complete an obstacle course collaboratively.

Leader selection control modalities require a human to directly control a robot or subset of robots within a swarm which then the other robots will follow or simulate its behavior. In [46] the authors proposed a control-Lyapunov based method of controlling multi-agent systems where a user controls a leader agent in a swarm. Other works have approached leader selection from a connectivity perspective [47], where the authors develop connectivity constraints on leader-follower networks with limited communication capabilities. The main benefits of this type of interaction are that the selection of the leader set of robots allows for increased forms of control that can be seen in single agent control such as teleoperation [48, 49], gesture control [50, 14] or other unique control techniques [51, 52]. Environmental control differs from this control paradigm in that instead of controlling individual agents or groups of agents in a swarm, users influence the environment and this results in behavior changes of the swarm. In [53] a multi-robot system is treated as a fluid and disturbances on this fluid generate motion on the dynamics of the robots in this system. Other works approach environmental influence through the use of potential fields [54, 55], artificial pheromones [56] or coverage control techniques [57, 58].

Parameter setting involves creating a set of parameters for a swarm allowing for an indirect influence on behaviors. Examples of this type of control modality can be seen in [16] where the authors influence the global characteristics of a swarm through parameters learned through evolutionary learning. Additionally, this type of control has been explored

for UAVs [59] where UAVs are assigned ”personalities” corresponding to parametric characteristics that determined how well mission plans are incorporated into its task allocation procedure.

In this thesis we focus on specification based control of multi-robot systems. This can be viewed as an extension of behavior selection as we generate high level goals for robotic systems that subsets or individual agents will execute. This type of robot control has been seen in [32, 60, 61]. Specification-based control allows users to define formal definitions for internal system behaviors with respect to an agent as well as the behaviors expected from agent interactions with its environment. In addition to this, specifications provide a logical formula and language that lends itself for ease of use when generating global goals for robotic systems.

2.2 UAV Motion Planning and Control

Before we address the multi-agent aspect of specification based planning using temporal logic, we first give a brief overview of motion planning and control for quadrotors in this section. Path planning for robotic systems has a wealth of literature related to the field [62, 63, 64]. Generally, these works are concerned with developing algorithms that convert high-level specifications of tasks from humans into low-level descriptions of robot control. Many fields address this general problem from control theoretic approaches that use system model information to generate feasible trajectories under system constraints [65, 66] to artificial intelligence algorithms which leverage heuristics, search techniques and state space information to make informed decisions on path planning [67, 68].

Using quadrotors has advantages for use in scenarios where space is limited due to their vertical takeoff and landing capabilities, efficient fuel source commonly found as battery technology and small and highly manueverable size amenable to locations that might be difficult for ground vehicles to explore. Thus, aspects of quadrotor planning and control that must be considered are the dynamical constraints and control capabilities that must be

addressed to generate satisfactory trajectories. Early works in quadrotor path planning used methods like virtual force fields to improve on 2D paths generated from graph searches [69]. Additional earlier works consider developing controllers designed to do trajectory tracking on the linearized dynamics of quadrotors [70, 71, 72].

Through the introduction of differential flatness theory, trajectory planning for quadrotors considers the full dynamical model of a quadrotor and aggressive trajectory manuevers become feasible [73, 20]. Differential flatness is a well known property of many dynamical systems [74, 75]. This property allows the expression of the full state and the input of the system as functions of the flat output and its time derivatives. In other words,

$$\boldsymbol{h} = g_h(\boldsymbol{s}, \boldsymbol{u}, \boldsymbol{u}^{(1)}, \ldots, \boldsymbol{u}^{(j)}) \tag{2.1}$$

is considered a flat output for j finite derivatives of input $\boldsymbol{u}$ if there exist two functions

$$\boldsymbol{s} = g_s(\boldsymbol{h}, \boldsymbol{h}^{(1)}, , \ldots, \boldsymbol{h}^{(l)}) \tag{2.2}$$

and

$$\boldsymbol{u} = g_u(\boldsymbol{h}, \boldsymbol{h}^{(1)}, \ldots, \boldsymbol{h}^{(m)}) \tag{2.3}$$

such that both the state ($\boldsymbol{s}$) and the input ($\boldsymbol{u}$) can be uniquely expressed as functions of the specified output and its l and m finite derivatives, respectively.

As shown in [5] and [76], for the quadrotor model described in Chapter 3, a possible choice of the flat output is $\boldsymbol{h} = [x, y, z, \psi]^T$ that allows the inference of the complete state ($\boldsymbol{s}$) and input ($\boldsymbol{u}$) with the appropriately defined functions. This means that by creating a path planner capable of generating trajectories $\boldsymbol{r} = [r_x, r_y, r_z, r_\psi]^T$ and its derivatives, it is possible to deduce the full state of the original nonlinear system relative to a particular reference. Many works that leverage differential flatness include those that consider fault

tolerance [77], safety constraints [78, 79] and path planning [80]. In this thesis we will show how we develop our planning framework to take advantage of differential flatness theory to develop trajectories capable of aggressive maneuvering by leveraging the full nonlinear dynamics of quadrotors.

2.3 Temporal Logic Control of Multi-Robot Systems

In this section, we will briefly overview the relevant literature on the use of temporal logic for the control of multi-robot systems. In a previous section, we introduced a variety of ways that humans can interact with multi-robot systems and introduced specification based control. Formal logic specifications, like LTL, provide an efficient and concise method for specifying and verifying correct behavior in dynamical systems and are well suited to human level interpretation and development due to its expressive syntax. LTL, linear temporal logic, was first introduced in the computer science literature [81] for specification of computer programs and guaranteeing safety and liveliness properties. LTL is a logic formalism suited for specifying linear time properties of a system [82, 22]. Defined over a finite set of atomic propositions $\Pi = \{\pi_0, \ldots, \pi_k\}$ where each proposition π_i maps from system state to true $(\top)$ or false $(\bot)$ and enables us to define a Boolean property of the state spaces, LTL formulas are formed with the following logic operators:

$$\phi := \top \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \mathbb{U} \phi_2 \tag{2.4}$$

LTL formulas are composed of the basic Boolean operators (e.g. conjunction $(\wedge)$ and negation $(\neg)$) and two temporal operators next $(\bigcirc)$ and until $(\mathbb{U})$. Formula $\bigcirc\phi$ evaluates as true at time t if ϕ holds at time step $t+1$. Formulas of the form $\phi_1 \mathbb{U} \phi_2$ evaluates as true at time t if ϕ_1 is true until a time step is reached where ϕ_2 becomes true. From these base operators of LTL we can construct the following higher-order temporal operators: $\Diamond$(eventually), $\Box$(always), and $\mathbb{W}$(weak until). The eventually formula $(\Diamond\phi)$ holds true at

time t if ϕ holds for some time $t_n \geq t$. The always formula ($\Box\phi$) holds true at t if ϕ holds at all times $t_n \geq t$. The weakly until formula ($\phi_1 \mathbb{W} \phi_2$) holds such that the occurrence of formula ϕ_2 does not need to be satisfied. Since its inception, the robotics and system engineering communities have adapted LTL for high-level motion planning and task allocation [83, 84, 85].

In the task allocation literature [86, 87] the problem of determining which robots should execute certain tasks in order to achieve a global system goal is addressed. These works seek to define a taxonomy of task allocation problems and the computational complexity associated with each problem. Prior works utilizing LTL for task allocation develop product automata consisting of environmental and agent systems are composed and satisfying sequences of states are found for robotic systems, defined as tasks, that satisfy that specification. These tasks are usually formulated as either motion primitives consisting of Lyapunov-based controllers [88], potential vector fields [89] or through symbolic control approaches like state space partitioning [18]. A work that is similar to ours, in the single agent case, is [90] where the authors proposed a sampling-based technique utilizing modified RRT* for agents to satisfy a global linear temporal logic specification. However, this work like many others relies on computationally expensive product automatons for large environments which are time consuming to generate [22]. In addition, each transition between tasks in these transition systems is often associated with action costs that are defined based on expert information about the system dynamics and environment. This approach is seen in many multi-robot systems using temporal logic control [91, 92] for task allocation. In this thesis, we approach the problem of multi-robot goal task allocation using the cross entropy method of stochastic optimization [93]. In this approach, we use cost functions and desired costs for each agent and allocate tasks based on which task best minimizes an individual agents cost function.

The cross-entropy method [94] is a form or importance sampling for rare events. It is used to draw samples from known probability distributions and minimize the cross-entropy

(or Kullback-Liebler distance) between the known distribution and a target distribution. Cross-entropy optimization has been used in many multi-agent goal satisfaction problems such as the max-cut problem [95], vehicle routing and other problems for which dynamic allocation of resources can be formulated as minimizing a cost according to a known desired cost and resources are drawn from a known distribution to estimate an unknown distribution [96]. In robotics, we have seen applications of cross-entropy and LTL for motion planning in the single agent case [97] and the multi-agent case [98]. Our work contributes to the use of cross entropy optimization for multi-UAV task allocation, leveraging the dynamics and environment constraints to assign tasks.

2.4 Learning Temporal Logic Specifications

In many applications of generating LTL specifications for multi-robot systems and single robot systems, the user must design a goal specification for the system. For novice users or large complex systems this may be infeasible to do without expert consultation. The field of LTL mining, deriving LTL specifications from user provided traces is extensive [35, 99, 100]. These works are commonly seen in computer science applications where researchers verify system performance through a set of trace logs provided to mining applications. Then, through a series of search and pruning techniques, Boolean propositions are composed to satisfy the set of traces provided. These types of solutions are often time intensive and can grow exponentially in the size of the proposition space. LTL learning differs from this in that models are used to learn from a set of features defined from a set of user provided traces the LTL specification that is satisfied by all the provided traces. Usually specifications are learned with respect to a set of templates (or property patterns) [101] that describe some desired atomic abstraction that the system is capable of performing. The final specification is then represented as a composition of these templates and aims to satisfy the set of traces provided. We can see this in works that use Bayesian inference for specification learning [39, 10] or through LTL learning problems modeled as Markov

Decision Processes [9, 29]. In [38] this problem is addressed using signal temporal logic (STL), a logic formalism that is expressive in spacial, temporal and logic relations, and uses a classifier algorithm to map traces to STL formulas.

In this thesis, we approach LTL learning through the use of support vector machines to classify traces as members of LTL templates and represent a desired specification as the composition of LTL templates generated from the templates chosen from the classifier for a given set of traces. This method improves upon prior LTL mining techniques by not performing an exhaustive search for specifications that are satisfied by the given set of traces offering consistency in performance as template sizes increase. In addition to this, for a tradeoff in perfect classification, we define a reduced LTL specification capable of defining system behavior over a reduced number of templates. We compare our algorithm to LTL mining techniques through run time analysis.

CHAPTER 3

SPECIFICATION-BASED MANEUVERING OF QUADROTORS THROUGH SUSPENDED HOOPS

Emerging unmanned vehicle applications such as farming [102] and reconnaissance [8] require navigating through a sequence of waypoints. With such applications, multiple objectives must be defined throughout the operation period of the vehicle. For example, a robot tasked with surveillance may be required to visit certain locations within a targeted area, report key information to users and periodically charge itself if its battery decreases below a threshold level. Despite a wealth of work on point-to-point motion planning[62, 63, 69], it is still challenging to provide end-to-end solutions for this waypoint problem on an actual platform that allows for reconfigurable waypoints and expressive task formulas.

Motivated by these applications and challenges, in this chapter, we study the problem of flying quadrotors through a sequence of hoops. This problem is dynamically challenging and using hoops provides a rich set of specifications for creating end-to-end planners. We first consider the problem using a set of oriented hoops, meaning they have a front and a back. The hoop flying problem can then be solved by a user directly specifying a desired sequence of hoops. We develop a sequence-of-hoops algorithm that ensures a quadrotor flies through this exact sequence when given as hoop/direction pairs by an external user. We also considers use cases that involve complex tasks where explicit sequences may be difficult to create.

For complex tasks, linear temporal logic formula can be used to characterize hoops rather than specifying them directly. For example, a user may specify "visit $hoop_0$ or $hoop_1$ before $hoop_2$" which can be represented by the LTL formula $\zeta = \neg hoop_2 \text{ U } (hoop_0 \vee hoop_1) \wedge \Diamond hoop_2$ or a user may indicate "avoid $hoop_1$ at all times" which can be captured in the formula $\zeta = \Box\neg hoop_1$. Work in [90] proposed a sampling-based technique for

agents to satisfy a global linear temporal logic specification, and [17] utilized a hybrid controller approach to design continuous trajectories using the dynamics of an autonomous system to satisfy LTL specifications. In addition to these prior works, we address the issue of autonomous systems satisfying global LTL specifications; however, our algorithm identifies more closely with the latter work as we consider the continuous dynamics of quadrotors in addition to considering tasks specified in linear temporal logic. Moreover, given a LTL specification, we model it as a Nondeterministic Büchi Automaton (NBA) and find a satisfying run [23]. The satisfying run corresponds to an infinite sequence of hoops that we parse through the planner as inputs. The planner then defines a trajectory via hand-picked control points around each hoop in the continuous space for quadrotors. We leverage the differentially flat dynamics of quadrotors [73, 5] to generate continuous trajectories using spline interpolation. With a solution to generating feasible trajectories, we turn to optimality.

This work follows [97] and [93] in applying cross-entropy optimization. We use cross entropy as a way to avoid discretizing the action space of the quadrotors, thus reducing computational complexity by not generating a transition system for the quadrotors. Cross-entropy optimization also improves upon feasible trajectories by minimizing over a user defined cost function.

Our work contributes to the application and development of controllers and algorithms designed to provide abstraction-free planning methods for autonomous vehicles subject to complex specifications given as a temporal logic formula. The main contributions of this chapter include:

- An end-to-end solution to the hoop flying problem via a planner given any explicitly labeled sequence of oriented hoops
- We utilize linear temporal logic without constructing a finite abstraction of the dynamics of the autonomous system to augment the expressive capabilities of our planner.

- We use cross-entropy optimization on nominal trajectories to optimize via user provided cost functions.

The chapter proceeds in the following manner. We present a brief overview of the quadrotor model and controller design in Section 3.1. We develop the mathematical foundation for the hoop flying problem and present three scenarios and approaches in Section 3.2. Finally, we discuss experimental results in Section 7.6 and present conclusions in Section 6.6.

3.1 Quadrotor Model and Controller

In this section, we give a brief overview of quadrotor dynamics and the controller used in experiments.

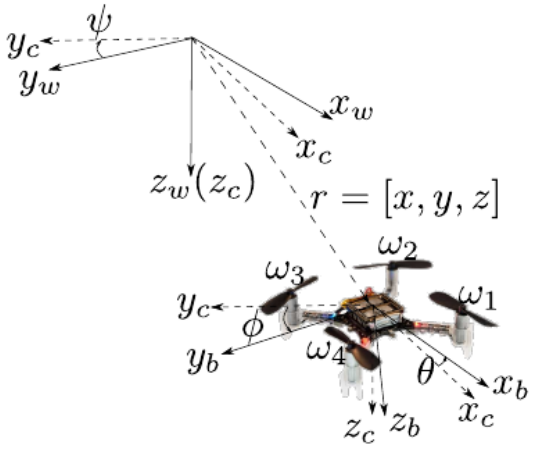


Figure 3.1: A quadrotor with respect to the world frame (F_w), intermediate frame (F_c) and body frame (F_b). Four motors ($\omega_{1:4}$) produce torques and thrust for the system.

The rotation matrix, $R(\varepsilon)$, that translates between the world frame (F_w) and body frame (F_b) is given by

$$R(\varepsilon) = \begin{bmatrix} c\theta c\psi & s\phi s\theta c\psi - c\phi s\psi & c\phi s\theta c\psi + s\phi s\psi \\ c\theta s\psi & s\phi s\theta s\psi + c\phi c\psi & c\phi s\theta s\psi - s\phi c\psi \\ -s\theta & s\phi c\theta & c\phi c\theta \end{bmatrix} \tag{3.1}$$

where $s\theta$ and $c\theta$ stand for $\sin(\theta)$ and $\cos(\theta)$, respectively. The angles θ, ψ, and ϕ are the angles between the axes of the quadrotor in the body frame and the axes of the world frame. The input (μ) to the system consists of $\mu = [f_z, \omega_{bw}^T]^T$ with f_z as the thrust and $\omega_{bw} = [\omega_x, \omega_y, \omega_z]^T$ as the body rotational rates of the quadrotor in the world frame. We use the quadrotor model from [20] to describe the dynamics that generate trajectories for quadrotors:

$$\ddot{r} = g z_w + \frac{1}{m} R(\varepsilon) z_w f_z \tag{3.2}$$

$$\dot{\varepsilon} = \Gamma(\varepsilon)\omega_{bw} = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi sc\theta & c\phi sc\theta \end{bmatrix} \omega_{bw}, \tag{3.3}$$

where $z_w = [0\,0\,1]^T$ is the z-direction vector for force in F_w and $sc\theta$ and $t\theta$ are $\sec(\theta)$ and $\tan(\theta)$, respectively. The position of the center of mass (r) in the world frame (F_w) is $r = [r_x, r_y, r_z]^T$; m, and g are the mass and acceleration of gravity, respectively. We represent the Euler angles as $\varepsilon = [\phi, \theta, \psi]^T$, and $\Gamma(\varepsilon)$ is the transformation matrix from body rotational rates in F_b to euler angles in F_w. These dynamics give the 12-dimensional state (ξ) of the quadrotors where $\xi = [r^T, \dot{r}^T, \theta, \phi, \psi, \omega_{bw}^T]^T$ and input (μ).

3.1.1 Controller

We define desired trajectories (η_d) as parametric curves in $\mathbb{R}^3$ that are three-times differentiable such that $\eta_d(t) \in C^3$. The state (ξ) and inputs (μ) are generated from trajectories using differential flatness [5, 75] to develop the controller. This allows us to represent

the entire state vector (ξ) of the quadrotor and its inputs (μ) as algebraic functions of the outputs. These outputs are called flat outputs, and are selected as $\eta = [r^T, \psi]^T$. Through differential flatness, we control the quadrotors using a feedforward term μ_{ff} adapted from the differentially flat outputs and a feedback term μ_{fb}. The μ_{ff} is derived by inverting (3.2) and (3.3) to get the feedforward thrust ($f_{z,ff}$) and body rotation rates ($\omega_{bw,ff}$). From [73],

$$f_{z,ff} = \gamma_(\eta, \dot{\eta}, \ddot{\eta}) = -m||\ddot{r} - gz_w||$$
$$\omega_{bw,ff} = \Gamma(\varepsilon)^{-1}\dot{\varepsilon} = \begin{bmatrix} 1 & 0 & -s\theta_d \\ 0 & c\phi_d & s\phi_d c\theta_d \\ 0 & -s\phi_d & c\phi_d c\theta_d \end{bmatrix} \dot{\varepsilon},$$

where $\theta_d = atan2(\beta_a,\ \beta_b), \phi_d = atan2(\beta_c, \sqrt{\beta_a^2 + \beta_b^2})$ and the d stands for desired. The functions β_a, β_b, and β_c are defined as: $\beta_a = -\ddot{x}_d \cos(\psi_d) - \ddot{y}_d \sin(\psi_d)$, $\beta_b = -\ddot{z}_d + g$ and $\beta_c = -\ddot{x}_d \sin(\psi_d) + \ddot{y}_d \cos(\psi_d)$.

For the feedback term μ_{fb}, the feedback thrust ($f_{z,fb}$) and body rotation rates ($\omega_{bw,fb}$) terms take the form,

$$f_{z,fb} = K_p\langle R(\varepsilon)z_w, r_d - r\rangle + K_d\langle R(\varepsilon)z_w, \dot{r}_d - \dot{r}\rangle,$$
$$\omega_{bw,fb} = K_p \begin{bmatrix} \phi_d - \phi \\ \theta_d - \theta \\ \psi_d - \psi \end{bmatrix} + K_d \begin{bmatrix} \dot{\phi}_d - \dot{\phi} \\ \dot{\theta}_d - \dot{\theta} \\ \dot{\psi}_d - \dot{\psi} \end{bmatrix} + K_q \begin{bmatrix} y_d - y \\ x_d - x \\ 0 \end{bmatrix}$$

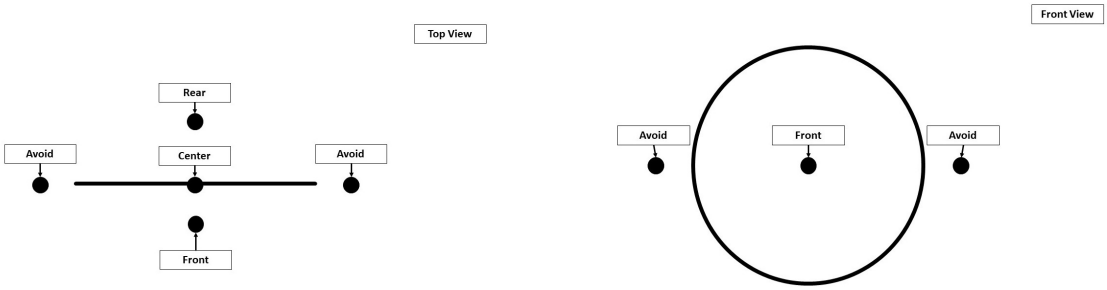
with gains K_p, K_d, and $K_q \in \mathbb{R}_{>0}$ where our total input to the system is $\mu = \mu_{ff} + \mu_{fb}$. We next define the hoop flying problem and our proposed approach.

3.2 Problem Formulation and Approach

Given a quadrotor with dynamics described in Section 3.1.1, we are interested in developing a fully autonomous planner capable of navigating through suspended hoops. We define

a hoop in $\mathbb{R}^3$, with the following definition:

Definition 1. *We define a hoop set* $\mathscr{H} = \{\mathscr{H}_0, \mathscr{H}_1, \ldots, \mathscr{H}_n\}$ *according to* $E_i(r) = \frac{(r_x - x_i)^2}{a^2} + \frac{(r_y - y_i)^2}{b^2} + \frac{(r_z - z_i)^2}{c^2}$ *and* $\mathscr{H}_i = \{r \in \mathbb{R}^3 \,|\, E_i(r) \leq 1\}$ *where*(r_x, r_y, r_z) *is the pose of the quadrotor,* (x_i, y_i, z_i) *is the position of a hoop (*$\mathscr{H}_i$*) with index i and* a, b*, and c are the x-radius, y-radius and z-radius of the hoops, respectively. We define these three constant radii* $(a, b, c) \in \mathbb{R}_{>0}$ *for the hoops to represent the volume covered by each hoop in our experiments and note they are equivalent for all hoops.*



(a) A top-down view of the orientation of control points for a hoop where the straight line depicted represents a hoop as seen from above. The five labeled dots represent the five control points (two avoid points, rear, center, and front) for each hoop.

(b) A front view of the hoop, depicted as a circle in the above image, with the five control points (two avoid points, rear, center and front). Note, the center and rear control points are occluded in this view.

Figure 3.2: The orientation of the control points around an example hoop. We define five control points as points in $\mathbb{R}^3$ at desired positions around hoops which are used as anchor points for trajectories generated between them.

Additionally, each hoop contains five control points (two avoid points, front, rear, and center) in $\mathbb{R}^3$, as illustrated in Figure 3.2. We leverage these control points as anchor points defining locations of interest near hoops and use them to guide trajectories between hoops. A motion plan is provided to the *sequence-of-hoops planner*. This motion plan is a sequence of hoops given by name and direction indicating which labeled side of the hoop the quadrotor must fly through. For example, to fly through "$hoop_0$" in the forward direction followed by "$hoop_1$" in the rear direction, the sequence would be G = '$hoop_0$F $hoop_1$R'. We now follow with the description of our sequence-of-hoops planner.

3.2.1 Flying through Hoops in a Given Sequence

In this section, we describe our sequence-of-hoops trajectory planner in Algorithm 1. In particular, we consider the following scenario:

Given a quadrotor and a set of pre-positioned labeled hoops, design a trajectory to fly through a specific sequence of hoops.

The algorithm receives as input the current pose of the robot (r), a set of n hoops ($\mathscr{H}$) (each defined by the control points previously mentioned), the input sequence ($\mathscr{G}$) of hoop/direction pairs, and the previous control points visited by the quadrotor ($c_{1:m}$). These control points are used to generate four distinct curves based on the distance of the current pose and future control points along with the direction of the quadrotor and generates segments (*segment*) via spline interpolation that joins control points together. We first define the current (*pos_curr*) and previous (*pos_prev*) pose which are found based on the order of the input sequence. In line 1, the past direction (dir_{past}) of the last control point transition is recovered from the list of previous control points.

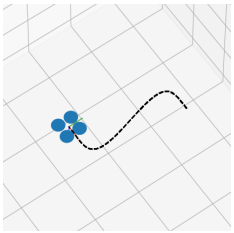
In line 2 - line 2 we define the next control point (*pos_next*) and the distance vector ($dist_{curr,next}$) to the next control point. In line 2, the function `NEXT_CONTROL_POINT` uses the current sequence to determine which control point on a labeled hoop is next. The function `DIRECTION` in line 2 gets the direction that the quadrotor is heading. This will inform our algorithm which type of trajectory, out of four pre-determined curves, will be chosen for a particular path segment. Starting at line 2, hoops that are in the state space but not in the input sequence ($\mathscr{G}$) are avoided using the `AVOID` function. The `AVOID` function receives as input the current position, a hoop from the list of hoops and the desired next position. If a hoop is between these two points, the function will choose the closest avoid control point of a particular hoop without crossing through that hoop and that resulting *segment* from the avoid control point to the current position will be added to the total trajectory $\eta(t)$.

Characterizing a Trajectory Segment: For the four types of trajectories that can be

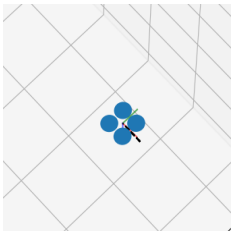
Algorithm 1 Sequence-of-Hoops Planner

input : pose of quadrotor r, hoops $\mathscr{H}$, input sequence $\mathscr{G}$, control_points_prev $c_{1:m}$
output: trajectory $\eta(t)$

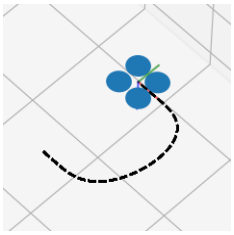
$pos\_curr \leftarrow r$
$pos\_prev \leftarrow c_m$
$dir_{past} \leftarrow$ DIRECTION(pos_curr, pos_prev)
for $i \leftarrow 0$ **to** $len(\mathscr{G})$ **do**
  $pos\_next \leftarrow$ NEXT_CONTROL_POINT$(\mathscr{G}_i)$
  $dist_{curr,next} \leftarrow$ DISTANCE(pos_curr, pos_next)
  $dir_{curr,next} \leftarrow$ DIRECTION(pos_curr, pos_next)
  for *hoop in* $\mathscr{H}$ **do**
    if *pos_next != hoop* **then**
      $segment \leftarrow$ AVOID$(pos_curr, hoop, pos_next)$
    end
  end
  if $dist^{yz}_{curr,next} == 0$ **then**
    $segment \leftarrow$ STRAIGHT(pos_curr, pos_prev)
  end
  if $dir_{curr,next} == dir_{past}$ **then**
    if $dist^{x}_{curr,next} \leq dist^{yz}_{curr,next}$ **then**
      $segment \leftarrow$ U_TURN$(dist_{curr,next})$
    else
      $segment \leftarrow$ S_CURVE$(dist_{curr,next})$
    end
  else
    $segment \leftarrow$ TURN$(dist_{curr,next})$
  end
  $pos\_prev \leftarrow pos\_curr$
  $pos\_curr \leftarrow$ RETURN_CONTROL_POINT$(segment)$
  $\eta(t) = \eta(t) + segment$
end
return $\eta(t)$



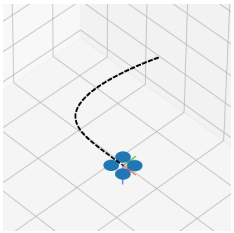
(a) S_CURVE: an arc produced when $dist_x > dist_{yz}$ and quadrotor is not changing direction



(b) STRAIGHT: a straight line curve generated in the x-direction when $dist_{yz}$ is zero



(c) U_TURN: curve produced when quadrotor is changing direction



(d) TURN: curve produced when $dist_x \geq dist_{yz}$ and quadrotor is not changing direction

Figure 3.3: Trajectory segments that are generated in the sequence-of-hoops planner.

generated, all are parameterized by time and are described by our algorithm in line 8 - line 18. We describe each function in detail here.

`STRAIGHT`: If $dist_{yz} == 0$, The function `STRAIGHT` in line 8 generates a trajectory segment length $||\eta_s(t)|| = |r_x^c - r_x^w|$ where r_x^c is the $x-component$ of the current pose and r_x^w indicates the $x-component$ of the previous pose.

`U_TURN`: A trajectory the length of two quarter arcs joined by a straight segment is returned by the `U_TURN` function in line 12 where $||\eta_s(t)|| = (dist_{yz} - dist_x) + (dist_x \cdot \frac{\pi}{2})$.

`S_CURVE`: In line 14, the length of the trajectory segment is created by first defining a right triangle leg (rt), such that $rt = \delta\sqrt{dist_x^2 - dist_{yz}^2}$. Then, theta ($\theta$) and the hypotenuse ($h$) of the triangle are defined as $\theta = atan2(dist_{yz}, dist_x)$ and $h = \frac{rt}{\sin\theta}$, respectively. The total length of the trajectory results in $||n_s(t)|| = 2(2h\theta)$ which generates two equal length tangent arcs or an "s-curve" in function `S_CURVE`.

`TURN`: If the direction is not the same as last, the trajectory length becomes a straight segment followed by a half-circle that changes the direction of the quadrotor such that $||\eta_s(t)|| = dist_x + (dist_y \cdot \frac{\pi}{2})$, the function `TURN` generates this segment.

In line 3.3 we show each type of trajectory segment that our planner can generate. The path lengths are then transformed into time segments via $t_s = \frac{||\eta_s(t)||}{v}$, where v is the desired speed. These time segments (t_s) along with the waypoints ($w_{1:m}$) given by the input sequence are used to generate smooth trajectories ($\eta(t)$) via spline interpolation.

Example: Using the sequence-of-hoops planner, we provide the following sequence $\mathscr{G}$ = '$hoop_0$F $hoop_1$F $hoop_2$F $hoop_1$R'. The provided sequence requires a quadrotor to navigate through four (4) unique hoop direction pairs in a specified order. In Figure 3.4, we provide an illustrated trajectory following the specified sequence passed to this algorithm. A path is planned through all desired hoops and the quadrotor navigates to a pre-specified final waypoint. In the depicted simulation, our trajectories are constrained with respect to predefined sequences. In the next section, we propose using the expressive capacity of LTL to enhance our algorithm. Moreover, we implement an improvement on the sequence-

of-hoops planner that can utilize LTL specifications to generate sequences that are not explicitly given by a user.

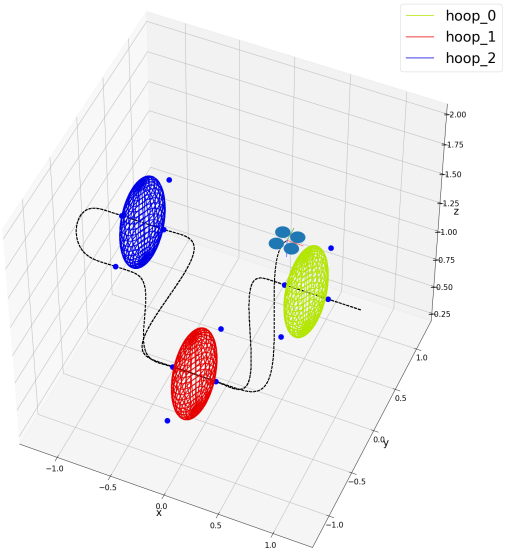


Figure 3.4: A sample sequence $\mathscr{G} =$ '$hoop_0$F $hoop_1$F $hoop_2$F $hoop_1$R' in simulation using the sequence-of-hoops planner.

3.2.2 Finding Satisfying Sequences Given an LTL Specification

In Section 3.2.1, we proposed a planner that can generate trajectories through explicitly defined sequences of hoops. However, suppose instead of satisfying explicit sequences, we utilized LTL as a method to convey high-level user input into trajectories that satisfy these specifications. Consider the following scenario: **Given a quadrotor and a set of pre-positioned labeled hoops, design a trajectory that satisfies an LTL specification.** For example, consider the following specification: "always ensure flying through $hoop_0$ implies $hoop_2$ is flown through before $hoop_1$ and eventually reach $hoop_1$". This specification can be represented by the LTL formula $\phi = \square(\lozenge hoop_0 \rightarrow \neg hoop_1 \cup hoop_2 \wedge \lozenge hoop_1)$.

We check that propositions $hoop_i$ are satisfied by mapping the hoop set ($\mathscr{H}_i$) in Definition 4.3, to the propositions through the labeling function k, where $k(r) = \{hoop_i \in \Pi : r \in \mathscr{H}_i\}$. In other words, we map the position of a quadrotor ($r \in \mathbb{R}^3$) to a set of corresponding hoop propositions. For example, $k(r) = \{hoop_0\}$ iff r belongs to $\mathscr{H}_0$. For continuous

quadrotor trajectories $\eta_c(t)$, we use a slight overload of notation for the following definition.

Definition 2. *Let us define k over a continuous trajectory where* $k(\eta_c) = \{hoop_{i_1}, hoop_{i_2}, hoop_{i_3}, \ldots, hoop_{i_j}\}$ *is the sequence of hoops visited by a quadrotor and* $hoop_i \in \Pi$ *and j indicates the* j^{th} *hoop in the sequence.*

This labeling function generates a sequence of propositions from the continuous trajectory $\eta_c(t)$ for $t \geq 0$. For example, a sequence could have the form $k(\eta_c) = \{hoop_0, hoop_1, hoop_0\}$. If the continuous trajectory $\eta_c(t)$ is created such that all propositions ($hoop_i$) generated from the trajectory satisfy an LTL formula ζ, then we have successfully found a trajectory that satisfies a given LTL formula. Therefore, the trajectory η_c satisfies the LTL formula ζ iff $k(\eta_c) \models \zeta$.

Generating Büchi Automata from LTL: After defining discrete propositions in continuous space, we can plan trajectories in the discrete space and map the trajectories back into the continuous domain using our sequence-of-hoops planner. From [23], any LTL formula can be represented as a Büchi Automaton, which are defined as:

Definition 3. *The tuple* $\mathscr{B} = (\mathscr{Q}, \Pi, \delta, \mathscr{Q}_0, \mathscr{F})$ *is a nondeterministic Büchi Automaton (NBA) where* $\mathscr{Q}$ *denotes a finite set of states,* Π *denotes the input alphabet,* $\delta : \mathscr{Q} \times \Pi \to 2^{\mathscr{Q}}$ *is the transition function,* $\mathscr{Q}_0 \subseteq \mathscr{Q}$ *represents the set of initial states, and* $\mathscr{F} \subseteq \mathscr{Q}$ *is the set of final states.*

A run $q = q_0 q_1 \ldots$ of a Büchi Automaton where $q_i \in \mathscr{Q}$ is an *accepting run* if $q_i \in \mathscr{F}$ for infinitely many indices. Associated with run q is a sequence of propositions ($\pi \in \Pi$) such that an infinite word $\sigma = \pi_0, \pi_1, \cdots \in \Pi$ is accepted if there is an accepting run for σ. Using efficient tools to translate LTL to nondeterministic Büchi Automaton (NBA) [103], we represent the NBA as a directed graph. We then search through the graph for runs σ of the prefix-suffix form $\sigma = (q_0, \ldots q_n)(q_{n+1}, \ldots)^{\omega}$ where the suffix portion of the run contains at least one state q_i within the accepting set. The n^{th} state indicates the last state

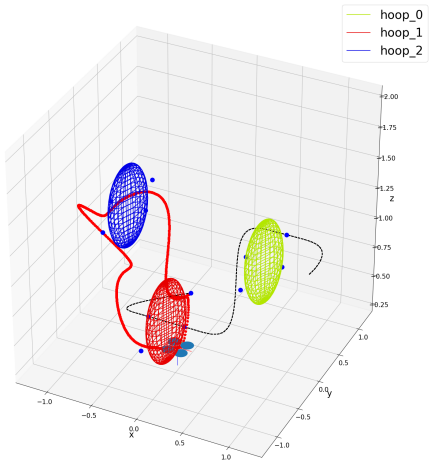


Figure 3.5: A satisfying run from the LTL specification ($\phi = \square(\lozenge hoop_0 \rightarrow \neg hoop_1 \cup hoop_2 \wedge \lozenge hoop_1)$). We show the prefix portion of the trajectory in black and the suffix portion in red. The suffix portion indicates the set of hoops that can be visited infinitely often.

in the prefix of the run. Once a satisfying run has been found, the satisfying word (i.e., a sequence of hoops) has been found and we apply our sequence-of-hoops planner to find a continuous trajectory.

Scenario Example: In regards to our proposed scenario, we augment the sequence-of-hoops planner to accept LTL specifications. We introduce a function `LTL_TO_SEQUENCE` which receives an LTL specification, generates an equivalent nondeterministic Büchi Automaton (NBA) [103] and searches for a satisfying sequence. In order to determine which direction (front or rear) the quadrotor must fly through the hoop from, we use the Euclidean distance between each proposed hoop to find the closest control point between two consecutive hoops in the sequence. The corresponding directions are appended to the hoops in the sequence and the planner executes as before. From the proposed scenario, we get the sequence: $\Sigma_{prop} = (hoop_1)(hoop_2)^{\omega}$ in prefix-suffix form. We note that Σ_{prop} is used to denote the accepting word for the NBA generated from our sample LTL specification and ω indicates the hoop, or set of hoops, that can be visited infinitely often during the execution of the algorithm. The resulting trajectory is shown in Figure 3.5.

3.3 Optimizing Sequences Using Cross-Entropy

Although the sequence-of-hoops planner is able to generate trajectories from user provided LTL specifications, the path is generally not optimal. We draw from the following scenario to motivate our problem:

Given a sequence-of-hoops planner for quadrotors, utilize the cross-entropy method to return trajectories that satisfy the sequence as well as minimize cost.

Use this method to guarantee an optimal trajectory with respect to a cost function $\mathscr{J}(r,u)$, parameterized by the robot pose, r, and its control input u. We use the cross-entropy method as a stochastic optimization technique for choosing trajectories according to our cost function, which we define to minimize the total length of the trajectory, i.e., $\mathscr{J}(r,u) = ||\eta(\cdot)||$.

In the following, a high-level overview of the cross-entropy method will be given followed by a proposed algorithm that further extends the augmented sequence-of-hoops planner in Section 3.2.2 to ensure optimality. For a more detailed description of the method see [94, 97].

The Cross Entropy Method: Cross-entropy optimization is a method of importance sampling for probabilistically rare events. The algorithm design for using cross-entropy with motion planning [93] can be generalized as the following:

1. Generate a set of sample trajectories (J) from a distribution and calculate cost $\mathscr{J}(r,u)$ for each trajectory

2. Update the distribution using a subset of samples (κ), until the sampling distribution converges to a desired cost (Σ) and delta function over the optimal trajectory

The subset of samples is defined as $\kappa = \rho J$, where $\rho = \{\rho \in \mathbb{R} : 10^{-1} \leq \rho < 0.3\}$ and J is ordered from least cost to greatest. While this method may not generate a globally optimal solution, due to it being a non-convex optimization method, the entire state space ($\mathscr{X}$) will

be explored during trajectory generation. In the next section, we describe our algorithm for minimizing the sequence-of-hoops trajectory using cross-entropy.

Optimizing the Sequence-of-Hoops Planner: From the augmented sequence-of-hoops planner in Section 3.2.2, we apply cross-entropy optimization to reduce the cost of the sampled trajectories once they are generated. Our algorithm is adapted from [97] with modifications on sampling initial means. We sample from the sequence-of-hoops planner to generate initial means to ensure that only the subset of the state space relevant to our hoop sequence is sampled.

Algorithm 2 Cross-Entropy Sequence-of-Hoops Algorithm

input : LTL formula ζ, hoop_propositions $i \rightarrow n$ $\mathscr{H}_{i:n}$, number of trajectories T, optimal cost Σ, elite set modifier ρ, sampling distribution $p(\mu_0, v)$, iteration number N

output: best_path $\eta(t)$

22 n := initial iteration number
μ_0 = Sequence-of-Hoops Planner(r, $\mathscr{H}_{i:n}$, ζ, $w_{1:m}$)
best_cost := ∞
while *best_cost* $>$ Σ *and* $n < N$ **do**
23 **for** i *in* T **do**
24 $path\_samples \rightarrow p(\cdot, v)$
$\eta(t) \rightarrow$ `PATH`$(path\_samples)$
$path\_check \rightarrow$ `LTL_PARSER`$(\zeta, \eta(t), \mathscr{H}_{i:n})$
if $path\_check$==$TRUE$ **then**
25 $sorted\_trajectories \rightarrow \eta_1(t) < \eta_2(t) \cdots < \eta_p(t)$
$best\_cost = sorted\_trajectories_0$
$elite\_set = \rho * sorted\_trajectories$
$p(\cdot, v) \rightarrow$ `UPDATE`$(sorted\_trajectories)$
26 **return** $\eta(t)$

Path samples are sampled from a multi-variate Gaussian distribution. The means (μ_0) are initialized to be n equidistant samples from the augmented Sequence-of-Hoops planner. As a result, we get path samples from line 24. In line 24 we define a `PATH` function that receives as input samples from the distribution $p(\mu_0, v)$ and generates a trajectory via spline interpolation. This trajectory is then checked in line 24 where it is monitored for inclusion in hoop sets ($\mathscr{H}$). The trajectory is then parsed in the syntax of the accepting set of hoops and checked for whether it satisfies the string received initially from the augmented planner

generated from the LTL formula (ζ). If this check returns $TRUE$ the trajectory is returned, otherwise a new trajectory is sampled. Trajectories are collected and sorted from best cost to worst cost and an elite set is chosen corresponding to a subset of trajectories (T). We then update the probability distribution using the elite subset of trajectories.

Scenario 4.3 Example: Using the cross-entropy sequence-of-hoops planner, we optimize over the LTL formula $\phi = \square(\Diamond hoop_0 \rightarrow \neg hoop_1 \cup hoop_2 \wedge \Diamond hoop_1)$ provided in line 3.2.2. In Figure 3.7, we show the optimized trajectory of the LTL formula with the constraint that the *each* trajectory segment (prefix and suffix) length should be less than 5 meters or $\mathscr{J}(r,u) \leq 5$. In Figure 3.6 we show the sampled paths over an iteration of the algorithm.

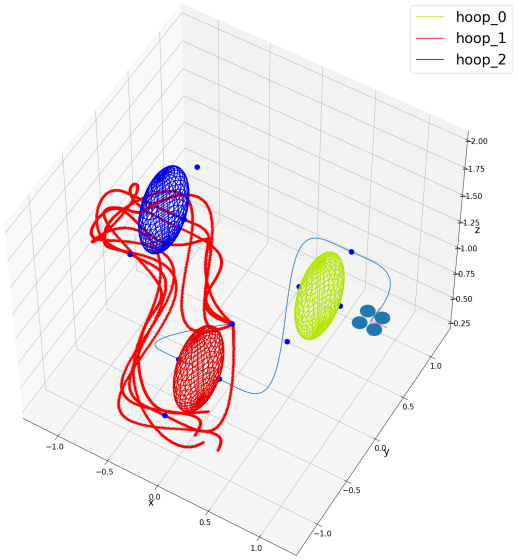


Figure 3.6: Here we show a series of trajectories generated from Algorithm 2. The prefix portion of the trajectory, represented in blue, satisfied the cost function $\mathscr{J} = ||\eta(t)||$ initially therefore, only one sample was needed. Several samples of the suffix portion, in red, of the trajectory were sampled before a satisfying trajectory was found.

3.4 Demonstration of Planner

We implement the planner on the Robotarium at Georgia Tech where we use Crazyflie 2.0 quadrotors. The Robotarium uses a Vicon Tracking system which records real-time

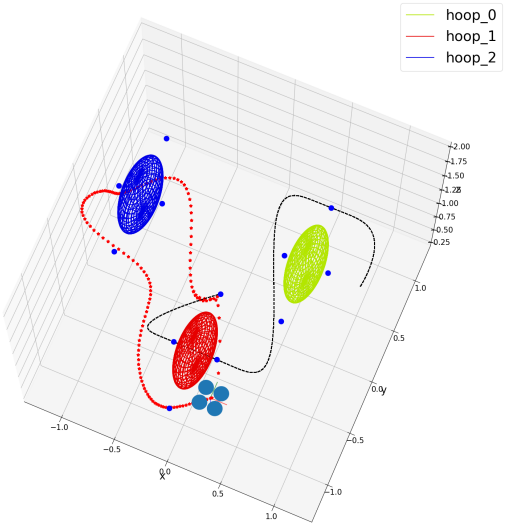


Figure 3.7: A simulation snapshot of the LTL cross-entropy sequence-of-hoops planner after a satisfying run. The optimized trajectory of is shown here where each trajectory segment must satisfy the cost constraint $\mathscr{J}(r,u) \leq 5$.

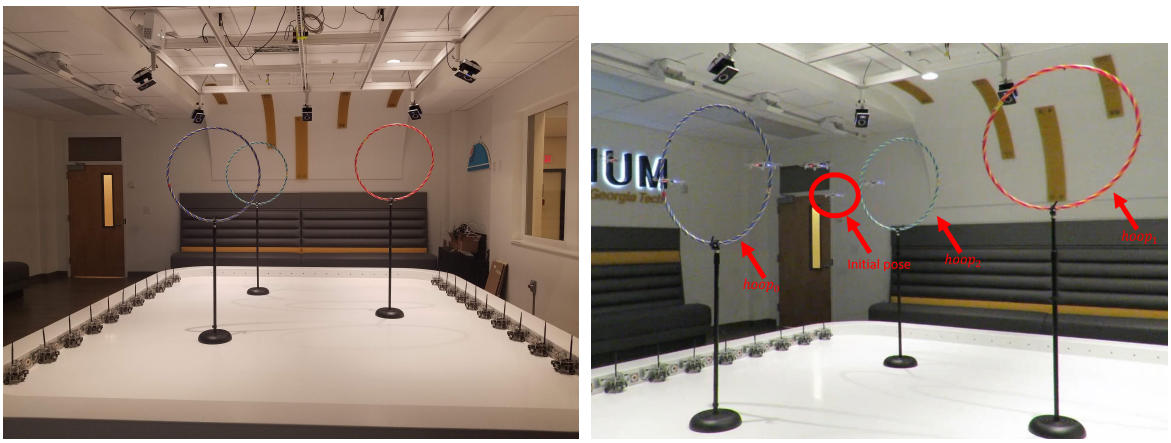
position of robots with a 100 Hz update rate. The algorithm was created in Python and sends control inputs to a PID controller in C++. Commands are sent via ROS messages to Crazyflies and a radio operating in the 2400 MHz range with a data rate of 2 Mbit/s sends these commands to the quadrotors. We orient hoops with vertical stands, as pictured in Figure 3.8a, in the Robotarium and mark them with Vicon tracking points to record the center of the hoops and generate the other control points.

Using the methods we have described in the previous sections, a user can specify either an explicit sequence of hoops or an LTL formula via a command line interface (CLI). Depending on the input, the sequence is then parsed through the sequence-of-hoops planner as a sequence of hoops or LTL formula and generates trajectories that maneuver a quadrotor through hoops satisfying that specification. The user is also given the option to give a minimizing cost as well. Using the cross-entropy LTL algorithm, trajectories are minimized via cost functions ($\mathscr{J}$) provided a priori.

In Figure 3.8b, a composite image of a quadrotor is shown completing a trajectory. We define the sequence ($\mathscr{G}$) for the sequence-of-hoops planner to execute as the same in Section 3.2.1. We use spline interpolation to generate trajectories between waypoints and

define a constant velocity of 0.45 m/s.

By restricting the planning to discrete hoop-to-hoop transitions we argue we have provided a potential solution to the hoop flying problem with our sequence-of-hoops planner. Using LTL, specifications as hoops can naturally be defined as discrete objects and enhances the range of specifications that the planner can satisfy. Leveraging the dynamics of quadrotors allows us to generate continuous trajectories, implement these trajectories on actual aerial robotic systems and provide applicable use cases for LTL with robotic systems.



(a) The hoops are mounted on adjustable stands and are tracked via Vicon markers.

(b) A composite image of the quadrotor executing the beginning of the automatically generated trajectory from sequence $\mathscr{G}$ = '0F 1F 2F 1R'.

Figure 3.8: Hoops are placed in the Robotarium and users can give sequences/LTL specifications via a command line interface (CLI).

3.5 Conclusion

Through the methods developed and implemented in this chapter, we design an end-to-end solution to the hoop flying problem. We show that given an explicitly labeled sequence of hoop/direction pairs the planner generates trajectories from four predefined trajectory segments to satisfy the sequence. We also show that for more complex specifications that may not be easily defined as sequences can be defined as LTL specifications and parsed through the planner as input. In addition, to ensure optimality, the cross-entropy optimiza-

tion method is utilized on nominal trajectories to optimize via user provided cost functions. In order to enhance the Robotarium user experience with quadrotors, we have implemented this as a novel interaction modality for external users to engage with aerial vehicles in the Robotarium.

CHAPTER 4
MULTI-AGENT TASK ORCHESTRATION VIA CROSS ENTROPY OPTIMIZATION

This chapter presents a task orchestration framework for multi-agent systems utilizing linear temporal logic (LTL) and cross entropy optimization, a stochastic optimization technique used for rare-event sampling. We define task orchestration as a combination of task decomposition, allocation and planning for a quadrotor or team of quadrotors given a high-level specification. Specifically, we consider tasks that are complex and consist of environment constraints, system constraints, or both, that must be satisfied. We first approached motion planning for the single agent case in Chapter 3 where transition systems for the environment allow tasks to be developed as linear temporal logic (LTL) specifications. Trajectories were generated via motion primitives for a single quadrotor and optimized via cross entropy to ensure optimal satisfaction of a cost function. We extend this work to the multi-agent case where a team of homogeneous quadrotors are considered to satisfy an LTL specification. In order to provide faster computations and initial cost-agnostic sampling, we formulate the online version of multi-agent task allocation via cross entropy for tasks specified in LTL specifications in Chapter 5. The results of this framework are verified in simulation and experimentally with a team of quadrotors.

In the previous chapter, we presented a planner capable of utilizing LTL to delegate high-level user specifications to a quadrotor. Through the introduction of motion primitives we simplify the path planning problem and introduce cross-entropy to optimize trajectory costs over a desired cost function. However, for large LTL specifications using a single agent may become infeasible. By using multiple agents, we can scale up the number of tasks a group of robots can perform and reduce the total time required to complete the tasks. Therefore, in this chapter we propose to use LTL as global task specifications, allowing

users to design global goals for multi-agent team execution. In addition to this, global goals enable scalability (i.e. goals are independent of the team size) and reduce cognitive load [1] on the designer as they do not have to assign each agent a specification. This type of interaction modality is easily adapted from temporal logic formula, in addition to providing formal guarantees for global specification satisfaction. In this chapter, we will give a brief overview of the transition systems used followed by an LTL decomposition framework that will allow for the decomposition of LTL specifications to a set of N agents.

Interaction with multi-agent systems often involves users requiring the satisfaction of a set of complex tasks. These tasks are delegated to the multi-agent system for a wide variety of reasons including: autonomous surveillance [42], search and rescue tasks [40] and environmental monitoring [104]. Often these tasks are defined as individual and environment constraints imposed on the system [32], [6]. These constraints must then be satisfied by a single agent or a multi-agent system while a main objective is reached according to an objective function or performance index. In this chapter, we formulate a system called *task orchestration*, defined as a composition of task decomposition, allocation and planning to provide an end-to-end framework for a team of quadrotors given constraints and an objective function. Specifically, users provide system constraints and objectives as linear temporal logic (LTL) specifications [23] and tasks are decomposed using a task decomposition framework and allocated according to an objective function for each agent in the multi-agent system. Finally, trajectories associated with a set of assigned tasks are generated for each agent.

We experimentally validate the application of the task orchestration framework through a fire-fighting quadrotors scenario. Consider a set of N quadrotors, capable of carrying water, surveying various locations and identifying resources within a predefined area. How does one dynamically allocate these quadrotors to different regions, extinguish fires and monitor their internal states in an efficient manner? Using the task orchestration framework, users give a desired global goal for the team of quadrotors to satisfy; the framework

then dynamically allocates tasks to each agent based on input cost and trajectory length, environment and agent constraints and plans trajectories for each agent. Experimentally, we define desired regions as hoops in the work space and generate an environment transition system to indicate how regions are connected. The internal state of a quadrotor is represented as the robot transition system.

With these formulations, we consider LTL as the global specification syntax users provide the task orchestration framework. Formal logic specifications, like LTL, provide an efficient and concise method for specifying and verifying correct behavior in dynamical systems and are well suited to human level interpretation and development due to its expressive syntax. High-level motion planning and task allocation using LTL allows for a diverse set of problems to be solved.

We addressed the issue of quadrotor high-level motion planning by creating motion primitives based on hoop/direction pairs in Chapter 3. In addition to providing a framework for high-level motion planning in the single agent case, we also reduce the time consuming and resource intensive process of product automaton generation for the multi-agent case. We do this in two ways: we leverage an LTL decomposition framework and introduce a method for cross-entropy optimization on trajectories sampled from multivariate distributions. The decomposition framework is from [34] and provides a theoretically sound method for decomposing a global LTL specification for an arbitrary number of agents. This bounds the size of the system product automaton to grow linearly with the number of agents as opposed to exponentially, as in most other works. We expand this work by introducing cross-entropy optimization which allows task delegation and switching to be based on the cost function of an individual agent. Cross-entropy optimization provides a method for stochastic optimization of trajectories by estimating rare-event probabilities (characterized as events that occur infrequently), which we associate with a desired cost and cost function. An alternative optimal sampling-based technique proposed in [105] leverages a probabilistically complete method of motion planning; however, it is not applied to multiple

goal satisfaction nor does it address how it can be used for multiple agents. However, many prior works exist that utilize cross-entropy optimization for multi-agent task allocation applications. It has been used in solving problems like the max-cut problem [95], vehicle routing problem [96] and other problems for which dynamic allocation of resources can be formulated as minimizing according to a known desired cost and resources are drawn from a known distribution in order to estimate an unknown distribution [97],[106]. This not only removes the need for an expert dependent action cost assignment but also allows general agent cost constraints to be defined within each agent that can leverage cross-entropy optimization for multi-agent task allocation.

In this chapter, we develop a framework for task orchestration in the multi-agent case. We presents a novel algorithm designed to sample trajectories and converge to a desired cost to determine the best robot from a team of robots for continued satisfaction of a goal specification and objective function. This algorithm advances the state-of-the art by introducing a formulation that allows users to design agent specific cost functions – for a homogeneous team of robots with equivalent dynamics – and dynamically allocate tasks over time while satisfying a global specification in addition to constraints of the environment or individual agent dynamics.

This chapter proceeds in the following manner. We develop the mathematical foundation for defining discrete transition systems and finite LTL in Section 6.2. A formal problem formulation is given in Section 4.2. We present the MTAC-E algorithm in Algorithm 4.3. Finally, we present a case study in simulation and provide experimental results in Section 4.4 and conclude in Section 6.6.

4.1 Defining Transition Systems

The framework for defining task decomposition [34] involves creating several state transition systems for a robotic system. From this discrete planning framework, we are able to decompose a product automaton containing multiple agents into independent tasks that can

be handled by each agent, while also satisfying a given goal specification. The definition of the robot transition system, $\mathscr{R}$, follows.

Definition 4. *The robot transition system is defined as a tuple* $\mathscr{R} = (S_{\mathscr{R}}, S_{\mathscr{R},0}, A_{\mathscr{R}}, \Pi_{\mathscr{R}}, \Lambda_{\mathscr{R}})$ *such that:*

- $S_{\mathscr{R}}$ *is a set of robot states*
- $S_{\mathscr{R},0} \subset S_{\mathscr{R}}$ *is the set of initial robot states*
- $A_{\mathscr{R}}$ *is a set of available robot actions*
- $\Pi_{\mathscr{R}}$ *is the set of robot propositions*
- $\Lambda_{\mathscr{R}} : S_{\mathscr{R}} \to 2^{\Pi_{\mathscr{R}}}$ *is a labeling function that assigns atomic propositions to states.*

The robot transition system captures the entire internal state of the robot and transitions are based on the actions, $A_{\mathscr{R}}$ available to the robot at each state. We next define the environment transition system $\mathscr{E}$ to capture the properties of the regions of interest for the agents.

Definition 5. *The environment transition system is defined as a tuple* $\mathscr{E} = (V_{\mathscr{E}}, E_{\mathscr{E}}, \Pi_{\mathscr{E}}, \Lambda_{\mathscr{E}})$ *such that:*

- $V_{\mathscr{E}}$ *is a set of environment vertices*
- $E_{\mathscr{E}}$ *is a set of edges between vertices where* $E_{\mathscr{E}} \subseteq V_{\mathscr{E}} \times V_{\mathscr{E}}$
- $\Pi_{\mathscr{E}}$ *is the set of environment propositions*
- $\Lambda_{\mathscr{E}} : S_{\mathscr{Y}} \to 2^{\Pi_{\mathscr{E}}}$ *is a labeling function that assigns atomic propositions to locations*

The product automaton $\mathscr{A}$ is used to define the internal state and external location of the agent throughout the planning space.

Definition 6. *The agent transition system is given as a product transition system* $\mathscr{A} = \mathscr{E} \otimes \mathscr{R} = (S_{\mathscr{A}}, S_{\mathscr{A},0}, A_{\mathscr{A}}, \Pi_{\mathscr{A}}, \Lambda_{\mathscr{A}})$ *such that:*

- $S_{\mathscr{A}} = V_{\mathscr{E}} \times S_{\mathscr{R}}$ *are the combined location and internal states of the agent*
- $S_{\mathscr{A},0} = \{(v, s_0) \in S_{\mathscr{A}} : s_0 \in S_{\mathscr{R},0}\}$ *is the set of initial agent states*
- $A_{\mathscr{A}} \subseteq S_{\mathscr{A}} \times S_{\mathscr{A}}$ *are the actions available to the agent*
- $\Pi_{\mathscr{A}} \subseteq \Pi_{\mathscr{E}} \times \Pi_{\mathscr{R}}$ *is the set of agent propositions*
- $\Lambda_{\mathscr{A}} : S_{\mathscr{A}} \to 2^{\Pi_{\mathscr{A}}}$ *is a labeling function that assigns atomic propositions to agent states*

In this definition, the set of actions $A_{\mathscr{A}}$ are available to a robot based on both its internal state and location in the environment. Additionally, the actions are restricted in that only actions that are available at states which satisfy the Boolean transition formula, $\xi : A_{\mathscr{A}} \to \psi$, are included. More formally,

$$A_{\mathscr{A}} = \{a = ((v,s),(v',s')) \in S_{\mathscr{A}} \times S_{\mathscr{A}} :$$
$$(v,v') \in E_{\mathscr{E}} \wedge (s,s') \in A_{\mathscr{R}} \wedge \Lambda_{\mathscr{A}}((v,s)) \models \xi(a)\}$$

Now that we have the agent automata defined for all agents, we can define the planning automaton $\mathscr{P}$ for the entire system.

Definition 7. *The planning automaton $\mathscr{P}$ is a product automaton of the NFA and agent transition system where $\mathscr{P} = \mathscr{F} \otimes \mathscr{A} = (S_{\mathscr{P}}, S_{0,\mathscr{P}}, A_{\mathscr{P}})$ such that:*

- $S_{\mathscr{P}} = Q \times S_{\mathscr{A}}$ *is the set of states*
- $S_{0,\mathscr{P}} = \{(q_0, s) \in S_{\mathscr{P}} : q_0 \in Q_0 \wedge s \in S_{\mathscr{A},0}\}$ *is the set of initial states*
- $A_{\mathscr{P}} = \{((q,s),(q',s')) \in S_{\mathscr{P}} \times S_{\mathscr{P}} : (s,s') \in A_{\mathscr{A}} \wedge \beta(s) \models \delta(q,q')\}$ *is the set of actions*

With the planning automaton $\mathscr{P}$, only sequences, σ – with propositions $\Pi_{\mathscr{A}}$ – that satisfy the LTL specification ϕ are accepted.

4.1.1 Decomposition Set

Given a multi-agent system with N agents, each represented according to the automata $\mathscr{P}^i$, we seek to decompose the global LTL specification ϕ such that parts of it can be assigned to the set of agents based on their cost functions. Moreover, using task decomposition, we wish to generate independent sequences of action/state pairs from $\mathscr{P}^i$ to satisfy ϕ where sequences are $s^i = s_0 a_0, \ldots, a_n s_n$. We give the following definition of finite LTL task decomposition.

Definition 8. *[34] Let $\mathscr{T}_i$ with $i \in \{1, \ldots, n\}$ be a set of finite LTL task specifications and σ_i denote any sequence such that $\sigma_i \models \mathscr{T}_i$. These tasks are called a decomposition of the finite LTL mission specification ϕ if and only if:*

$$\sigma_{j_1} \ldots \sigma_{j_i} \ldots \sigma_{j_n} \models \phi \tag{4.1}$$

for all permutations of $j_i \in \{1, \ldots, n\}$ and all respective sequences σ_i.

From this definition of decomposition we can create the decomposition set $\mathscr{D} \subseteq \mathscr{Q}$ of the NFA $\mathscr{F}$ developed from ϕ. This set contains all states q for which the pair of tasks $\mathscr{T}_1^q, \mathscr{T}_2^q$, where q is a state in the decomposition set $\mathscr{D}$, define a valid decomposition. For a proof of this property, we refer the reader to [34].

We use this to avoid generating a large product automaton of the transition system of agents and automaton representation of the finite LTL specification. This greatly reduces the computational complexity usually encountered with systems involving a large number of agents. We define team product automata, $\mathscr{T}$, with the following definition.

Definition 9. *The team model automaton $\mathscr{T}$ is a union of the N local product automatons $\mathscr{P}^i$ with $i \in \{1, \ldots, N\}$ where the tuple is $\mathscr{T} = (S_{\mathscr{T}}, S_{0,\mathscr{T}}, A_{\mathscr{T}}, F_{\mathscr{T}})$ such that:*

- $S_{\mathscr{T}} = \{(r,q,s) : r \in \{1, \ldots, N\}, (q,s) \in S^i_{\mathscr{P}}\}$ *is the set of states*

- $S_{0,\mathscr{T}} = \{(r,q,s) \in S_{\mathscr{T}} : r = 1\}$ *is the set of initial states, with r being a randomly assigned initial agent*
- $A_{\mathscr{T}} = \bigcup_i A^i_{\mathscr{D}} \cup \zeta$ *is the set of actions, including the switch transitions Z*
- $F_{\mathscr{T}}$ *is the set of accepting final states*

Switch transitions, Z, allow our algorithm to select a new agent within the product automaton to complete the satisfaction of the specification.

Definition 10. *The switch transitions in* $\mathscr{T}$ *are given by* $Z \subset S_{\mathscr{T}} \times S_{\mathscr{T}}$. *A transition* $\zeta = ((r_s,q_s,s_s),(r_t,q_t,s_t)) \in Z$ *if and only if [34]:*

- $r_s \neq r_t$*: the agents are different*
- $q_s = q_t$*: the progress of the NFA is preserved*
- $r_t = r_s + 1$*: A new agent is selected*
- $s_t = s^{r_t}_{0,\mathscr{A}}$*: The new state is the initial state of a new agent*
- $q_s \in \mathscr{D}$*: the state is in the decomposition set of the NFA*

4.2 Problem Formulation

With discrete transition systems defined for a homogeneous team of agents and a decomposition framework, we turn to our problem formulation.

Problem: For a given set of homogeneous agents, distribute tasks among these agents considering discrete agent transition systems with unknown action costs. Distribute these tasks while minimizing individual agent cost functions $f_i(\cdot)$, given by the operator before execution, for agents $i,\ldots,N$.

We demonstrate this problem as a firefighting quadrotor scenario in Section 4.4. In this problem, we designate N quadrotors, each defined by discrete product automata as our set of homogeneous agents, with no action costs to transition between states. The

swarm of robots is given the global task of surveying goal locations within the state space, acquiring water and transporting it to the desired location while obeying the constraints of the environment. We solve this problem using the MTAC-E algorithm proposed below.

4.3 MTAC-E Algorithm

Solution: We propose the Multi-Agent Task Allocation Cross-Entropy (MTAC-E) Algorithm to delegate tasks to a set of agents. Previously, we defined a decomposition framework in Section 6.2; given we have designed cost functions for each agent in the problem, we need a way to find optimal trajectories by minimizing these cost functions. To find the associated minimized costs, we propose using cross-entropy optimization. In this framework, we use cost functions optimized via cross-entropy as opposed to static actions costs defined at discrete state transitions. This additional flexibility in problem design allows operators to minimize over individual agent cost functions and use generalized functions for entire agent trajectories when the cost to perform an action is unknown. We present a brief overview of cross entropy and our algorithm in the following sections.

4.3.1 Cross-Entropy Optimization

Cross-entropy optimization is a method of importance sampling for probabilistically rare events. The algorithm design for using cross-entropy with motion planning [93] can be generalized as the following:

1. Generate a set of sample trajectories (J) from a distribution $p(\cdot, x)$ and calculate cost $\mathscr{J}(\cdot)$ for each trajectory
2. Update the distribution, p, using a subset of samples (κ), until the sampling distribution converges to a desired cost (λ) and delta function over the optimal trajectory

The subset of sampled trajectories with the lowest cost (i.e. $\kappa \subset J$) is defined such that $|\kappa| = \rho|J|$, where typically $10^{-1} \leq \rho < 0.3$. This subset is known as an "elite set" and provides

a new sampling space to generate the distribution p. In this work, we sample trajectories according to a multi-variate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$ such that $\mu = [\mu_0, \ldots, \mu_n]^T$ for n equally spaced points along the set of sampled trajectory. The covariance matrices, $\Sigma = [\Sigma_0, \ldots, \Sigma_n]^T$ form an $nm \times m$ matrix with Σ_i initially set to the identity matrix, I. Expectation-Maximization [107] is used to update the means and covariances for the newly sampled trajectories. In the next section, we describe the multi-agent task allocation cross-entropy (MTAC-E) algorithm.

4.3.2 Algorithm

The algorithm developed in this chapter, provided in pseudocode format in Algorithm 3, can be described in four steps:

1. given the initial state of the agent product automaton, find the cost of transitioning to the next state using cross-entropy and the cost function assigned to the agent
2. if this state is contained in the decomposition set, check all other agent cost functions and
3. if an agent has a lower total cost, switch to this agent for the remainder of the algorithm or until a new switch is determined
4. this process in continued until the end state is found and corresponding trajectories are returned to all agents for execution.

The algorithm receives as input the team product automaton, $\mathscr{T}$, the decomposition set, $\mathscr{D}$, an optimal cost for each agent to minimize towards, λ, the elite set modifier, ρ, an initial sampling distribution, p and the number of times to iterate the sampling procedure, K. In Line 27, the initial state, p_i, the agent of p_i ,α_i, and the current sequence of states visited by agent i, $sequences_i$, are initialized. We recall Definition 9 of the team product automaton in this framework such that via a standard BFS search, once the state $p \in final\_states(\mathscr{T})$ is found and a sequence is generated that reaches this state, the LTL specification is satisfied.

The cross-entropy optimization technique in Line 29 is utilized in the function `cost_to_go`. An initial probability distribution is provided for each agent with initial means and variances. Also, elite set modifiers (ρ), an optimal cost (λ), and a bounding maximum iteration number (K) are supplied as input. The function samples from the given distribution and iterates until either the cost function has been met or the maximum iterations has been exceeded and returns the trajectories for each agent ($\eta_i(t), \ldots, \eta_n(t)$).

For states in the decomposition set ($\mathscr{D}$), a cost is calculated from each in Line 30 and if one of the costs is less than the current agent's cost ($cost_i$) the agents are swapped and the new agent j continues the remainder of the sequence until the next switch transition occurs. By switching the agents at decomposition states, this algorithm optimizes the individual task function of each agent via cross-entropy and optimally allocates tasks to minimize the total cost of an individual agent. This algorithm will return a set of trajectories $\mathscr{N}$ with each agents individual trajectory $\eta_i(t)$.

4.3.3 Complexity

We give a brief overview of the complexity of the algorithm and compare it to other methods for task allocation using temporal logic. Size analysis to search through LTL automata for satisfying sequences is well-known [23]. Generally, a trajectory, η can be checked if it satisfies the automata $\mathscr{A}_\phi$ in $O(|\eta| \cdot |\mathscr{A}_\phi|)$, denoting a bilinear complexity in the length of the trajectory and in the size of the automata. Leveraging task decomposition, the size of our team automaton, $\mathscr{T}$ is much smaller than one created via a product automata (i.e. $\mathscr{A}_{prod} = \mathscr{P}_i \otimes \mathscr{P}_{i+1}, \ldots, \mathscr{P}_{N-1} \otimes \mathscr{P}_N$), where N is the number of agents. In our work, we check trajectories for membership in an agent planning automaton, $\mathscr{P}_i$, which is equivalent to the number of NFA states $\mathscr{F}$ times the number of agent states $\mathscr{A}$ or $|\mathscr{P}_i| = |\mathscr{F}| \cdot |S_\mathscr{A}|$ unlike automata produced by constructing a product where $|\mathscr{A}_{prod}| = |\mathscr{F}| \cdot |S_\mathscr{A}|^N$, thus $|\mathscr{P}_i| \ll |\mathscr{A}_{prod}|$. Due to the checking of N agents in our framework, our algorithm can check trajectories with complexity of $O(N \cdot (|\eta| \cdot |\mathscr{P}_i|))$. Recall, that product automata

Algorithm 3 MTAC-E Algorithm

input : product automaton $\mathscr{T}$, decomposition set $\mathscr{D}$, optimal cost λ, elite set modifier ρ, sampling distribution $p(\mu_0, v)$, iteration number K

output: set of trajectories $\mathscr{N}$

p_i := initial_state($\mathscr{T}$)

$\alpha_i \rightarrow p_i$:= agent i in initial state

$sequences_i$:= sequence of states visited by agent i

$p \rightarrow p_i$:= set p to initial state

while $p \notin$ *final_states*($\mathscr{T}$) **do**

  for q *in neighbors(p)* **do**

    $\eta_i(t)$, $cost_i \rightarrow$ `cost_to_go`(α_i, q, $sequences_i$, λ, $p(\cdot, v)$, K, ρ)

      if $q \in \mathscr{D}$ **then**

        $\eta_{j:n}(t)$, $cost_{j:n} \rightarrow$ `cost_to_go`($\alpha_{j:n}$, q, $sequences_{j:n}$,λ, $p(\cdot, v)$, K, ρ)

      end

      if $cost_{j:n} < cost_i$ **then**

        $\alpha_i \rightarrow \alpha_j$

        $p \rightarrow p_j$

        $sequences_j = sequences_j + sequences_{j,p \rightarrow q}$

      end

      else

        $sequences_i = sequences_i + sequences_{i,p \rightarrow q}$

      end

  end

end

$\mathscr{N} = \{\eta_1(t), \ldots, \eta_n(t)\}$

return $\mathscr{N}$

have states that grow exponentially with the number of agents therefore, due to our algorithm being linear in the number of agents, N, we show our algorithm is far more scalable than other methods utilizing product automata for task allocation. In addition to this, the runtime of the MTAC-E Algorithm, while heavily dependent on cost function choice and size of planning automaton, is ~ 300 seconds for the task allocation of three agents.

4.4 Case Study: Fire Fighting Drones

We motivate the application of Algorithm 3 with a firefighting UAVs scenario. For example, each agent may be a fire-fighting autonomous aircraft capable of collecting water, extinguishing fires and surveying goal locations. These UAVs are given the following global goal: "*eventually* visit LOC_1 and LOC_2 and *always* ensure visiting $SMOKE$ implies $CARRYING$". Using LTL, this specification can be represented as $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge \Box(SMOKE \implies CARRYING)$.

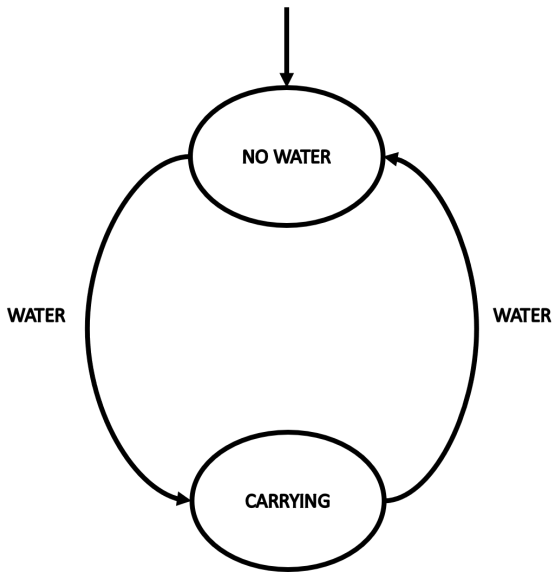


Figure 4.1: A transition system for a single agent which describes the internal state of a robot ($\mathscr{R}_i$). All robots start at the initial state 'NO WATER' and a location-based transition is used to determine when to transition to the 'CARRYING' state. If a robot is in the environment state that satisfies the 'WATER' proposition, the robot can transition to the 'CARRYING' state.

According to our discrete planning framework, we define the internal state of the robot

using Definition 4 where our robot is represented by a two state transition system with a transition denoted by whether it has visited the water location in the environment. A robot transitioning from the 'NO WATER' state to the 'CARRYING' state indicates the 'WATER' proposition was true in the environment during that transition. In Figure 4.1 we represent the discrete internal transition system of robot *i* as ($\mathscr{R}_i$).

The environment transition system, Figure 4.2, is represented by a set of nodes corresponding to states with adjacent nodes in the graph representing neighbors for potential paths through the state space. In simulation and experiment, we represent each node as an ellipsoid in $\mathbb{R}^3$. Formally, these ellipsoids have the following form:

Definition 11. *The environment proposition set* $\Pi_{\mathscr{E}} = \{\mathscr{E}_i, \ldots, \mathscr{E}_n\}$ *is defined as:*

$$E_j(r) = \frac{(r_x - x_j)^2}{a^2} + \frac{(r_y - y_j)^2}{b^2} + \frac{(r_z - z_j)^2}{c^2} \tag{4.2}$$

$$\mathscr{E}_j = \{r \in \mathbb{R}^3 \quad | \quad E_j(r) \leq 1\} \tag{4.3}$$

where(r_x, r_y, r_z) *is the pose of the quadrotor,* (x_j, y_j, z_j) *is the position of a region of interest* ($\mathscr{E}_j$) *with index* j *and* a, b, *and* c *are the x-radius, y-radius and z-radius of the regions, respectively. We define these three constant radii* $(a, b, c) \in \mathbb{R}_{>0}$ *for the regions to represent the volume covered by each ellipsoid in our experiments and note they are equivalent for all ellipsoids.*

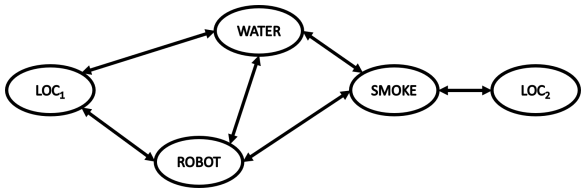


Figure 4.2: The environment transition system where each state indicates a desired region of interest. The initial state of the environment is the 'ROBOT' state. In the fire fighting example, LOC_2, cannot be reached unless the quadrotor passes through the SMOKE region.

Using this definition, discrete transitions are identified when the relative position of a

quadrotor transitions inside any of the regions of interest defined in the state space. In our case study, the environment proposition set is $\Pi_{\mathscr{E}} = \{WATER, SMOKE, LOC_1, LOC_2\}$. By taking the product we can generate the full agent automaton for each agent i such that $\mathscr{A}_i = \mathscr{E} \otimes \mathscr{R}_i$ shown in Figure 4.3. Following the standard procedure for developing automata for robotic systems we generate a NFA from the finite-LTL specification and take the product with $\mathscr{A}_i$ for each agent to get P, an automaton that only accepts runs that satisfy the LTL specification and agent transition system.

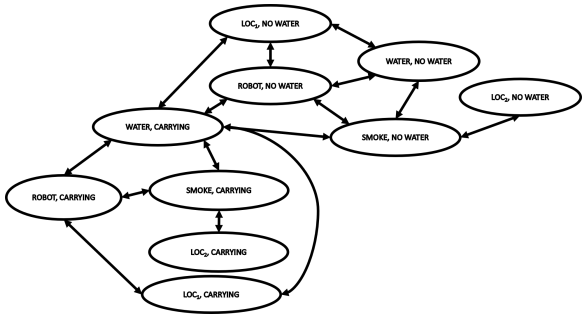


Figure 4.3: The full agent transition system for a quadrotor. Transitions to the 'CARRYING' state can only be fulfilled once the agent has retrieved water from the environment node.

4.4.1 Deriving the Control Input

Utilizing the differentially flat dynamics [5] of the quadrotors, we represent the inputs and outputs of the system as algebraic function of chosen flat outputs and their derivatives. From this property, trajectories can be generated by leveraging the nonlinear dynamics of the quadrotors. This leads to the ability to plan smooth trajectories that are three-times continuously differentiable functions, $\eta(t) \in C^3$, in the output space that can be converted back analytically into feasible trajectories for the full state of the quadrotors. We utilize a virtual input $u \in \mathbb{R}^3$ from [20] that controls a chain of integrator dynamics for the differentially flat outputs of the system.

4.4.2 Simulation

We apply Algorithm 3 to the disjoint product of the n agents P automaton, $\mathscr{T} = P_i \cup \cdots \cup P_n$. In order to generate trajectories from the given specifications we utilize a custom sequence planner that uses pre-selected trajectories based on a quadrotor's position and speed relative to a labeled location (e.g. an ellipsoid's location and generate splines between ellipsoids). After the initial trajectory for a given sequence is plotted, we use cross-entropy optimization to minimize that trajectory over the cost function $\mathscr{J} = \int_0^T \eta(\tau) + u(\tau) d\tau$.

The MTAC-E Algorithm samples trajectories from an unknown distribution that minimizes the cost function, $\mathscr{J}$, which is a function of the path length, $\eta(t)$ and the control input, $u(t) = \ddot{r}$ where $r = [x, y, z]^T \in \mathbb{R}^3$, the position of the center of mass of the robot. We set $\lambda = 0$, indicating a desired optimal cost for each agent of minimal trajectory length and cost. In general, a trade-off is made between picking a reasonable λ and algorithm run-time, which is why limiting the number of iterations is desirable.

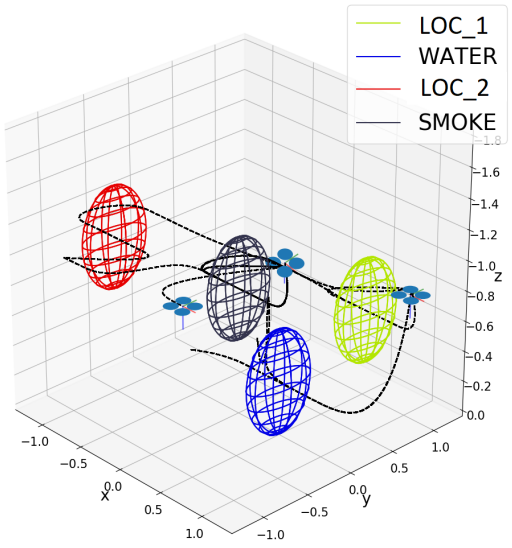


Figure 4.4: Three quadrotors during a simulated fire fighting mission. The entire team is given the specification $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge \Box(SMOKE \implies CARRYING)$. Each quadrotor is considered during the iteration through the product automaton of the system, switches to another quadrotor are considered when the cost is beneficial for the team.

Results are shown in Figure 4.4 where three quadrotors are shown satisfying the LTL

formula ϕ. The results sequences are $quad_0 = \{WATER \quad SMOKE \quad LOC_2\}$, $quad_1 = \{SMOKE\}$ and $quad_2 = \{WATER \quad LOC_1\}$ which results in a satisfying sequence for the entire input specification. The returned sequences are one of many satisfying sequences returned by a search over the team automaton, $\mathscr{T}$, and additional constraints in the graph can modify which sequences are returned.

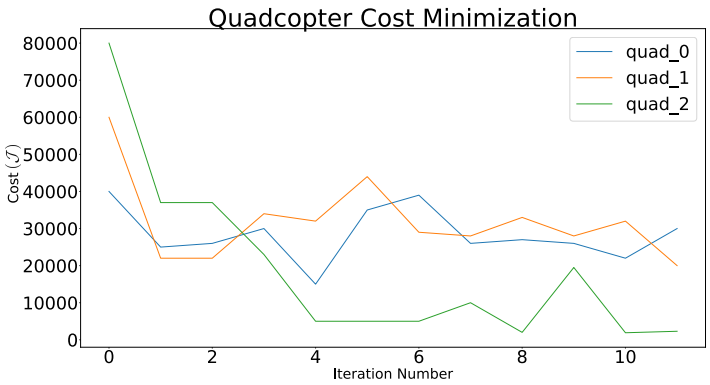


Figure 4.5: The MTAC-E Algorithm iterates 12 times over a subset of trajectories and produces the trajectory with the lowest cost after all iterations. Here, we show the algorithm evaluating which quadrotor should transition to LOC_2. This calculation is formulated in our cost function where we minimize the distance traveled and input to system. Each quadrotor executes the MTAC-E optimization and after all quadrotors have completed the algorithm, the quadrotor with the lowest cost is selected to complete that task, in this example $quad_2$ is chosen.

4.4.3 Experimental Results

The MTAC-E Algorithm is implemented on the Robotarium at Georgia Tech where we use Crazyflie 2.0 quadrotors. The Robotarium uses a Vicon Tracking system which records real-time position of robots with a 100 Hz update rate. The algorithm was created in Python and sends control inputs to a PID controller in C++. Commands are sent via ROS messages to Crazyflies and a radio operating in the 2400 MHz range with a data rate of 2 Mbit/s sends these commands to the quadrotors.

We use hoops with vertical stands to represent regions of interest, characterized by ellipsoids, as pictured in Figure 4.6, and mark them with Vicon tracking points to record

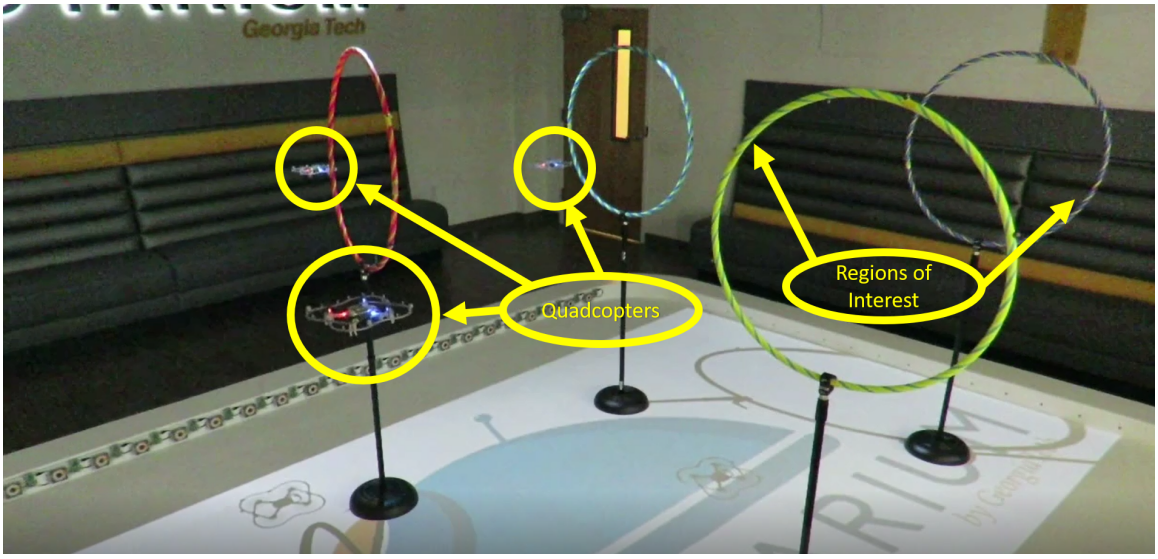


Figure 4.6: In the experimental case study of the firefighting quadrotors, three quadrotors are chosen to execute the global task specification. The entire team is given the specification $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge \Box(SMOKE \implies CARRYING)$ and each are given a cost function to minimize. Regions of interest are represented as ellipsoids and hoops on stands are used in the experiment.

the center of the hoops and generate the proposition sets. In this experiment, we utilize three quadrotors and deploy them with the same LTL specification used in Section 4.4.2. Based on the cost constraint $\mathscr{J}$, defined previously as functions of position and control input, tasks are assigned to agents based on proximity requirements to goal locations, control input constraints and prior tasks executed. Resulting from this problem, the quadrotors are allocated tasks in the following sequences $quad_0 = \{WATER \quad LOC_1 \quad SMOKE\}$, $quad_1 = \{SMOKE \quad WATER \quad LOC_2 \quad SMOKE\}$, and $quad_2 = \{WATER \quad SMOKE\}$. This experiment demonstrates the practical use of quadrotors in a real world scenario, delegating tasks to the agents in an optimal fashion using the MTAC-E algorithm.

4.5 Conclusions

In conclusion, we have developed a novel method for multi-agent task allocation using cross-entropy motivated by task switching for decomposed sequences of tasks. This method allows users to design global specifications to multi-agent systems where exact action costs for agents are not known to the user a priori but a known distribution can be approximated

through a cost function. In addition, cost functions can be defined for individual agents depending on agent specific constraints. We show that this algorithm is scalable and flexible in system and environment constraint satisfaction through an operator chosen cost function. We showcase the efficacy of the algorithm both in simulation and in experiment under a scenario that demands satisfaction of environmental constraints and system constraints while optimizing a cost function designed to minimize individual agent trajectories and inputs.

CHAPTER 5
ONLINE MULTI-AGENT TASK ALLOCATION

The MTAC-E algorithm developed in Chapter 4 allowed the use of a stochastic optimization technique for task allocation of a multi-agent system. In addition to this, we utilize cost functions to assign agents a task based on the individual constraints of an agent and its environment. However, this algorithm was developed in an offline manner, meaning it must run to completion before tasks can be delegated to a set of agents. This is time consuming and detrimental for time critical task assignment scenarios. The offline algorithm also considers optimal costs defined a-priori by an expert individual. This assumption makes it difficult for users to utilize this algorithm without prior information of the desired cost function, agent and environment that a multi-agent system will operate in. We solve both these issues by developing an online method of cross-entropy, modified to operate over multi-dimensional probability distributions. This method allows fast generation of trajectories in real-time and an expert independent method of optimal cost generation.

In addition to this, the online multi-agent task allocation framework we propose provides fast calculations of optimal agent assignments and greatly reduces the task allocation computation time. One work that is close to ours is that of [98], where the authors use adaptive cross-entropy for task assignment for UAV formations, optimized over a global cost function. However, these authors do not consider path planning in their problem formulation and only consider task allocation for simulated vehicles.

This chapter improves on the multi-agent task allocation algorithm by modifying our framework to consider online task allocation, providing fast updates to desired trajectories allocated to a team of robots. This novel approach to task allocation provides online trajectory sampling from a known distribution and iteratively updates based on the desired quantile of the multivariate distribution associated with a desired trajectory. By providing

this online methodology we combine the reduction in product automata size from task decomposition with an expert independent framework for choosing optimal costs and a fast way to dynamically allocate tasks to a set of agents which scales linearly.

This chapter extends the work from [108] and [106] by significantly expanding the capabilities of the previously developed multi-agent task allocation algorithm via the online cross-entropy optimization method developed in this chapter. In particular, we present a novel characterization of online cross-entropy over multivariate distributions and develop a general, task orchestration framework that utilizes this stochastic optimization methodology for online task assignments to individual agents in a multi-agent system. We compare this online result to the prior offline work, and empirically show a significant decrease in overall cost and execution time. In addition to this, we validate the online approach for task orchestration of multi-agent systems on a particular fire-fighting quadrotors scenario.

5.1 Online Cross Entropy

In order to apply an online version of the MTAC-E algorithm, we consider the trajectory $\mathscr{X}_t$ sampled at time t. The trajectory $\mathscr{X}_t = (\mathscr{X}_{1t}, \ldots, \mathscr{X}_{nt})$ is a vector of i.i.d. random variables $\mathscr{X}_{it}$ drawn from known multivariate Gaussian distributions $\mathscr{N}(\mu, \Sigma)$. Similar to [109], we consider a sample, $\mathscr{X}_{N_e}$, elite if at position $\rho \cdot |X|$, with $\rho > 0$ and $|X|$ the cardinality of the set of trajectories drawn at time t, it belongs to the subset of ordered trajectories $\{\mathscr{X}_0 < \mathscr{X}_1 < \cdots < \mathscr{X}_{\rho \cdot |X|}\}$. At each time step, this sample is chosen based on if its corresponding cost $f(\mathscr{X}_{N_e}) \geq \lambda_t$ where λ_t is the elite threshold. The threshold (λ_t) either increases, decreases or stays constant according to four cases depending on where the new sample and the dropout sample belong in the set of N trajectories:

1. New sample ($\mathscr{X}_{[N_e+1]}$) and dropout sample ($\mathscr{X}_{[N_e]}$) are elite, this is a rare event with probability ρ and as such has a small probability of occurring. In this event, the threshold, λ and threshold position stays the same and does not change.

2. New sample is elite but dropout sample is not elite. In this case, the threshold value will increase by the difference in cost between the new sample and the sample at the end of the elite set such that $\lambda_{t+1} = \lambda + f(\mathscr{X}_{[N_e+1]}) - f(\mathscr{X}_{[N_e]})$. This increase in λ is attributed to "expanding" the search of trajectories with similar costs to gain membership to the elite set.

3. New sample and dropout are not elite. The threshold will stay the same.

4. New sample is not elite and dropout sample is elite. In this case, the threshold value will decrease since the thresholding may be too high for samples to be considered elite. Thus, the threshold is lowered to $\lambda_{t+1} = \lambda + f(\mathscr{X}_{[N_e-1]}) - f(\mathscr{X}_{[N_e]})$.

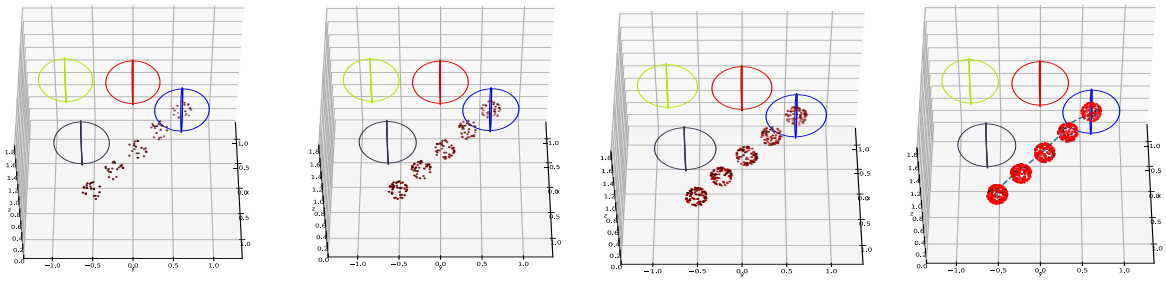
Let $\Delta_t = E(f(\mathscr{X}_{[N_e+1]}) - f(\mathscr{X}_{[N_e]})|\mathscr{G}_t)$ be the update step for the threshold where $\mathscr{G}_t$ is the σ-algebra of all known random outcomes up to time t and $f(\mathscr{X}_{[N_e]})$ as the trajectory cost measured at the elite set threshold. Given that the sample positions within the set of trajectories are distributed uniformly,

$$\begin{aligned}
&E(\lambda_{t+1}|\mathscr{G}_t, \text{new sample is elite}) \\
&= \lambda_t + P(\text{case 1}) \cdot E(f(\mathscr{X}_{[N_e+1]}) - f(\mathscr{X}_{[N_e]})|\mathscr{G}_t) + P(\text{case 2}) \cdot 0 \\
&= \lambda_t + (1-\rho) \cdot \Delta_t.
\end{aligned}$$

Likewise, for the non-elite sample case

$$\begin{aligned}
&E(\lambda_{t+1}|\mathscr{G}_t, \text{new sample is not elite}) \\
&= \lambda_t + P(\text{case 3}) \cdot 0 + P(\text{case 4}) \cdot E(f(\mathscr{X}_{[N_e-1]}) - f(\mathscr{X}_{[N_e]})|\mathscr{G}_t) \\
&= \lambda_t - \rho \cdot \Delta_t.
\end{aligned}$$

The update step is equivalent to the product of the average difference between samples and the difference between $\mathscr{X}_t$ evaluated at the ρ^{th} probability and 1 sample above it. In order



(a) t=2.5 (b) t=5 (c) t=7.5 (d) t=10

Figure 5.1: Samples drawn from the inverse distribution $\mathbb{F}^{-1}$ are shown in the above plot. Sample means μ are measured and shifted by the quantity given by the product of the covariance of $\mathcal{N}(\mu,\Sigma)$ and the standard normal inverse function Φ^{-1} evaluated at the ρ^{th} quantile. In these plots, sample means are measured via the straight line distance between a quadrotor and the desired hoop, the covariance matrix is $\Sigma = 0.05 \cdot \mathscr{I}$ and $\rho = 0.05$. At each iteration, new samples are drawn from the quantile $\mathbb{F}^{-1}$, here we show samples, corresponding to points drawn at the ρ^{th} quantile, drawn at different times during the runtime of the sampling algorithm. The final iteration at t=10 contains the trajectory the quadrotor follows.

to create an update step for trajectories sampled from a Gaussian distribution, we develop a Gaussian approximation for multivariate distributions. Prior works that utilize online cross entropy for task allocation used an update step based on uniform or univariate Gaussian distributions, relying on one dimensional samples. Our work extends this by formulating an update step for multivariate Gaussian distributions in the following section.

5.2 Developing the Update Step

The update step for samples from multivariate distributions will be a $n \times m$ matrix corresponding to n samples from $\mathscr{X}_t$, with each sample a m-dimensional vector. Consider a continuous update step that measures samples from normal distributions according to the desired ρ^{th} percentile. From the inversion principle [110]:

Fact 5.2.1. *Let Φ be the cumulative distribution function on $\mathbb{R}^n$ with the inverse Φ^{-1} defined as*

$$\Phi^{-1}(p) = inf\{x \in \mathbb{R}^n : \Phi(x) \leq p, 0 < p < 1\}. \tag{5.1}$$

1. *If U is a uniform [0, 1] random variable then $\Phi^{-1}(U)$ has distribution Φ.*

2. *If X has distribution Φ, then $\Phi(X)$ is distributed uniformly on [0, 1].*

Proof. The first statement can be verified through

$P(\Phi^{-1}(U) \leq x) = P(\inf\{y \in \mathbb{R}^n : \Phi(y) \leq U\} \leq x) = P(U \leq \Phi(x)) = \Phi(x).$

The second statement is a result of the following relationship

$P(\Phi(X) \leq u) = P(X \leq \Phi^{-1}(u)) = \Phi(\Phi^{-1}(u)) = u$ □

The normal distribution we sample from is not a standard distribution, therefore our inverse function Φ^{-1} is

$$\mathbb{F}^{-1}(\rho) = \mu + \Sigma \cdot \Phi^{-1}(\rho) \tag{5.2}$$

where ρ is the desired percentile of $\mathcal{N}(\mu, \Sigma)$ and Σ is the covariance matrix. In [111], the average of two samples from a normal distribution is $2\sigma/\sqrt{\pi}$, extending to the multivariate distribution case, we form $\Delta_t = E|\mathscr{X}_1 - \mathscr{X}_2| \cdot \delta_{\mathbb{F}^{-1}(\rho)}$ as the difference between the quantile functions for a particular sample weighted by the expectation of two samples from a normal distribution with known parameters such that

$$\delta_{\mathbb{F}^{-1}(\rho)} = \frac{1}{2}\mathbb{F}^{-1}(1-\rho+\frac{1}{N}) - \mathbb{F}^{-1}(1-\rho) \tag{5.3}$$

$$\Delta_t = \frac{1}{\sqrt{\pi}} \cdot \Sigma[\mathbb{F}^{-1}(1-\rho+\frac{1}{N}) - \mathbb{F}^{-1}(1-\rho)]. \tag{5.4}$$

This formulation of Δ_t is a matrix and a scalar form is desirable for doing computations on λ, another scalar. We propose the following definition to achieve a scalar approximation of delta, $\Delta_{\tilde{t}}$.

Fact 5.2.2. *Let Δ_t be the multivariate update step for threshold cost, λ. The scalar approx-*

imation of Δ_t is formulated as

$$\Delta_{\tilde{t}} = \min_i \sum_{j=1}^{m} |\Delta_{ij}| \in \mathbb{R} \tag{5.5}$$

minimizing the sum of quantile differences and returning the smallest difference between quantile samples.

Using $\Delta_{\tilde{t}}$ we update the desired threshold λ_t in real-time as samples are acquired. However, the quantile function $\Phi(x)$ is not easily acquired. In the next subsection, we define how to sample from a hyperellipsoid to generate uniformly sampled points from the ρ^{th} quantile of $\Phi(x)$.

5.2.1 Generating Quantile Function for Multivariate Gaussian

We begin by defining our problem as finding the contour line of a hyperellipsoid derived from the parameters of $\mathcal{N}(\mu, \Sigma)$.

Fact 5.2.3. *Let* $\mathbf{Y}$ *be a sample from the surface of the hyperellipsoids with the following form*

$$(\mathbf{Y} - \overrightarrow{\mu})^T \Sigma^{-1} (\mathbf{Y} - \overrightarrow{\mu}) \leq c^2$$

Where c is the desired distance of $\mathbf{Y}$ *from mean* $\overrightarrow{\mu}$*. A semi-axis of the hyperellipsoid is* $\sigma_i = \pm c\sqrt{\gamma_i} v_i$ *and contains 100(1-*ρ*)% of the sampling distribution and* $c^2 = \chi^2_{\rho,\alpha}$*, the chi-squared distribution for* α *degrees of freedom measured at its* ρ^{th} *value.*

Proof. The eigendecomposition of the covariance matrix is $\Sigma = VDV^T$, with V as the eigenvector matrix and D as the diagonal matrix of eigenvalues γ_i. We find the square root of $\Sigma = VS^{1/2}V^T$. $\Sigma = \Lambda\Lambda^T$ where $\Lambda = VD^{1/2}$. The matrix Λ scaled by a factor c results in

$\Lambda^* = c\Lambda$ and likewise $\Sigma^* = c^2\Lambda\Lambda^T = c^2VD^{1/2}D^{1/2}V^T = c^2\Sigma$ which determines the contour of the distribution considered by the sampled vector $\mathbf{Y}$. □

With our desired distance value, c, we project points sampled from a uniform hyperspheroid, R, onto the covariance Σ, shifted by the mean vector, μ. The points from hyperspheroid, R are sampled using the following algorithm [112] and we develop the following algorithm for sampling from quantile functions of multivariate distributions:

Algorithm 4 Quantile Sampling

Input : desired value of quantile function ρ, covariance matrix Σ, mean vector $\vec{\mu}$, degrees of freedom α, number of samples m

Output: point sampled from ρ^{th} quantile,Y

41 $c^2 \rightarrow \mathscr{X}^2_{\rho,\alpha}$
$c \rightarrow \sqrt{\mathscr{X}^2_{\rho,\alpha}}$
$\Sigma^{1/2} = VD^{1/2}V^T$
$\Lambda = c\Sigma^{1/2}$

42 $X_{nz\times m} \sim \mathscr{N}(0,1)$:
$r_{ss} = \sqrt{\sum_{i=0}^{i=m}(X^2_{nz\times i})}$
$K_x = \mathbf{1}_{nz\times 1} \otimes r_{ss}$
$X_{nz\times m} = X_{nz\times m}/K_x$
$z = \vec{\mu} \cdot \mathbf{1}_{1\times\alpha}$
$Y = X^T_{nz\times m} \cdot \Lambda + z^T$

We provide a brief description of Algorithm 4 here. We begin with sampling the chi-squared distribution for a desired number of samples m at percentile ρ with α degrees of freedom at line 41. Followed by this, we find the square root of the received value and store it as c. We find the eigendecomposition of the covariance and its corresponding square root followed by the definition of Λ at line 41. Afterwards we follow the developments from [112] to sample from a hyperellipsoid. In line 42 we take the square root of the sum of squares of samples from a normal distribution sampled m times. The matrix $X_{nz\times m}$ results in points uniformly distributed on a hypersphere. Because we wish to access the values of this hypersphere stretched by $c\Sigma$ and centered on mean z, we apply the linear equation

in line 42 to find points sampled from the ρ^{th} quantile, as shown in Figure 5.1 over 5 selected means. This proposed method allows us to sample from a desired percentile of our sampling distribution for multivariate Gaussians. Desired trajectories corresponding to elite samples can be drawn from this sampling function and we use this sampling function to generate Δ_t at each iteration.

5.3 Online Cross Entropy Sampling Algorithm

In Algorithm 5 from lines 43 - 45 we check if the run is the first run of the algorithm, if true, initial means μ_t are generated based on the current position of the quadrotor (r) and position of the hoop ($\mathscr{H}$). In addition to this we generate covariances (Σ), a randomly generated value for the desired cost (λ) and Δ_t is generated from the `Quantile_Sampling` function where we assume that the samples are generated from a distribution with N trajectories. If the run is not the initial run, we use the previous μ, λ, and Δ_t. Following this, we perform the matrix operation from Equation 5.5 on Δ_t. We sample a trajectory $\mathscr{X}_t$ from a normal distribution by interpolating between path samples drawn from the distribution using the means and covariances previously acquired. If the cost function $f(\mathscr{X}_t)$ and the trajectory satisfy the constraint in line 51 then λ_t is increased. Otherwise, λ is decreased conditional on the case that the trajectory goes through the hoop but does not satisfy the cost constraint. Finally, Δ_t is updated and we return the pose in $\mathscr{X}_t$ that corresponds to time t.

5.4 Update Δ_t Threshold

Our value for incrementing the desired threshold, γ, can also be modified with an exponential factor $\beta = ae^{-bt}$ with $a, b > 0$. The delta threshold is updated at each time step t such that $\Delta_{\tilde{t+1}} = (1-\beta)\Delta_{\tilde{t}} + \beta|f(x_t) - f(x_{t+1})|$. Using this update formulation, initially, $\Delta_{\tilde{t}}$ weights information from the costs between samples as more important than prior $\Delta_{\tilde{t}}$ values. This is beneficial for us since our samples are drawn uniformly from a quantile

Algorithm 5 Online Cross Entropy

Input : robot position r, desired hoop $\mathscr{H}$, desired value of quantile function ρ, previous cost $f(x)_{t-1}$, previous time t_{t-1}, previous lambda λ_{t-1} previous delta Δ_{t-1}, previous means μ_{t-1}, previous covariancesΣ_{t-1}, previous samples X_{t-1}, number of total trajectories N

Output: updated desired state p_t, means μ_t, covariances Σ_t, path samples X_t, lambda λ_t, cost $f(x)_t$, delta Δ_t

if *initial run is True* **then**

  $\mu_t \rightarrow$ `generate_means`$(r, \mathscr{H})$
  $\Sigma_t \rightarrow$ `generate_initial_covariances`()
  $\lambda_t \rightarrow$ `random_generator`()

  $\Delta_t \rightarrow \frac{\Sigma}{\sqrt{\pi}} \cdot$ `Quantile_Sampling`$(1-\rho+\frac{1}{N}, \mu_t, \Sigma_t)$ - `Quantile_Sampling`$(1-\rho, \mu_t, \Sigma_t)$

end

else

  $\mu_t \rightarrow \mu_{t-1}$
  $\lambda_t \rightarrow \lambda_{t-1}$
  $\Delta_t \rightarrow \frac{\Sigma}{\sqrt{\pi}} \cdot$ `Quantile_Sampling`$(1-\rho+\frac{1}{N}, \mu_t, \Sigma_t)$ - `Quantile_Sampling`$(1-\rho, \mu_t, \Sigma_t)$

end

$\Delta_{\tilde{t}} \rightarrow \Delta_t$
$\mathscr{X}_t \sim \mathscr{N}(\mu, \Sigma)$

if $f(\mathscr{X})_t \geq \lambda_{t-1}$ *and* `goes_through_hoop`$(\mathscr{X}_t)$ *is True* **then**

  $\lambda_t \rightarrow \lambda_t + (1-\rho) \cdot \Delta_{\tilde{t}}$
  $\mu_t, \Sigma_t \rightarrow$ `MLE`$(\mathscr{X}_t, \ldots, \mathscr{X}_{t-1})$

end

else if `goes_through_hoop`$(\mathscr{X}_t)$ *is True* **then**

  $\lambda_t \rightarrow \lambda_t - \rho \cdot \Delta_{\tilde{t}}$

end

$\Delta_t \rightarrow (1-\beta)\Delta_{\tilde{t}-1} + \beta\Delta_{\tilde{t}}|(f(\mathscr{X}_t) - f(\mathscr{X}_{t-1})|$
$p_t \rightarrow \mathscr{X}_t(t)$
return p_t, μ_t, Σ_t, λ_t, $f(x)_t$, Δ_t

function whose initial samples may not be reliable but allows a sufficient search of the cost function space of nearby samples. In Figure 5.2 we show how varying values of a and

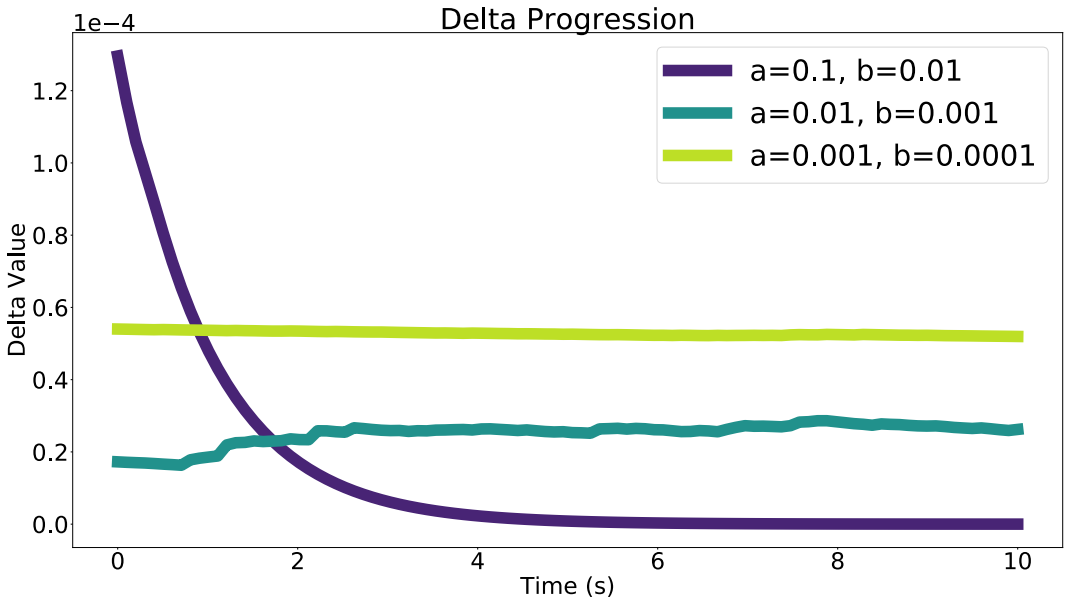


Figure 5.2: The time evolution of $\Delta_{\tilde{t}}$ for varying values of a and b. From the figure, a greater value of a and b indicate a greater dependence on the absolute error in sample trajectory costs $f(\mathscr{X})$ initially. This causes a greater change in magnitude of the step size of $\Delta_{\tilde{t}}$ than in smaller values of a and b and may reduce the sensitivity of the algorithm to desirable values of $\Delta_{\tilde{t}}$. We observe through empirical tests that values near $a = 0.01$ and $b = 0.001$ provide a reasonable trade-off between utilizing the trajectory costs and prior step size Δ_t for updating $\Delta_{\tilde{t}}$.

b affect the convergence of $\Delta_{\tilde{t}}$. In this figure, we plot the change in $\Delta_{\tilde{t}}$ as the quadrotor runs the online cross entropy algorithm to converge to a desired hoop in 10 seconds. Initial $\Delta_{\tilde{t}}$ update values can vary due to dependence on $f(\mathscr{X}_t)$, so instead we focus on the convergence rate of each plot. We can see that for $a = 0.1$ and $b = 0.01$ as time progresses, $\Delta_{\tilde{t}}$ quickly converges to a value close to zero as $\Delta_{\tilde{t}}$ moves from reliance on relative difference in measured cost to previous $\Delta_{\tilde{t}}$ values. On the other extreme, for $a = 0.001$ and $b = 0.0001$, $\Delta_{\tilde{t}}$ does not move far from its initial value and stays near the value it eventually converges to during the run time. For all experiments, we chose $a = 0.01$ and $b = 0.001$ due to the values allowing for initial cost consideration, followed by slow convergence to values closer to expected quantile differences.

5.5 Online MTAC-E Algorithm

Algorithm 6 Online MTAC-E Algorithm

Input : current robot poses $r_{i,\ldots,n}$ time t, LTL spec ϕ

Output: new poses $r_{i,\ldots,n}$, flag for specification satisfaction $satisfied\_ltl$

58 $satisfied\_ltl$ is `False`
 if *first iteration* **then**
59 | `satisfying_run` $\rightarrow$ `generate_satisfying_run(`ϕ`)`
60 **end**
61 **for** *q in* `satisfying_run` **do**
62 | **if** *q is satisfied* **then**
63 | | **continue**
64 | **end**
65 | **if** *all q satisfied* ***OR*** *all agents assigned* **then**
66 | | **if** *all nodes satisfied* **then**
67 | | | $satisfied\_ltl$ is `True`
68 | | **end**
69 | | **break**
70 | **end**
71 | **if** $q \in \mathscr{D}$ **then**
72 | | $r_{i,\ldots,n-1} \rightarrow$ `switch_condition(`$r_{i,\ldots,n-1}$`,` $f_{i,\ldots,n-1}(x,u)$`, t)`
73 | **end**
74 | **else**
75 | | $r_j \rightarrow$ `online_cross_entropy(...)`
76 | **end**
77 **end**
78 **return** $r_{i,\ldots,n}$, $satisfied\_ltl$

The Online MTAC-E algorithm presented in Algorithm 6 proceeds in the following section. The current robot poses for n robots is given, the time t, and the desired LTL specification (ϕ) to be satisfied. If the algorithm is in its first iteration we generate the sequence of propositions that satisfy the specification, otherwise we store this desired sequence for future iterations. Then for each node, in lines 61 - 70 a series of checks are done to verify if the node has been satisfied or if all the nodes have been satisfied. In addition to this, we track if each agent has had a chance to be assigned a proposition, when all agents have an assignment, the desired states are sent to each robot. If the robots are in the decomposition set $\mathscr{D}$ we consider the switch condition where each agents current state, desired trajectory

to node q, and the cost function $f_i(x,u)$ associated with a particular agent to assign the node to the agent with the lowest cost. If the node is not in the decomposition set, the online cross entropy algorithm executes for a single agent. Due to the node not being in the decomposition set, once an agent is assigned to the set $\bar{D} = \{q \in \bar{\mathscr{D}} | q_i \neq q_0 \wedge (q_i, q_{i+1}) \in \mathscr{E}_{\bar{\mathscr{D}}}\}$ it may not transition to the decomposition set for the remainder of the run. The set of states p are returned for each robot agent where each state is the chain of integrators state referred to in Section **??**. We have now formally generated an online method for task allocation using the stochastic optimization technique, cross entropy, for quadrotors. However, now we verify the algorithm against the offline version followed by experimental results.

5.5.1 Comparison to Offline MTAC-E

In this section we compare the efficiency defined as the individual agent costs and run time performance of the online MTAC-E with the offline version. In Figure 5.3, we compare the sum of agent costs for a predefined number of agents ranging from 1- 5. For each run, we assign the agents to satisfy the LTL specification . From the figure, we see that the online method is more efficient at overall task assignment as total agent costs are greatly reduced compared to the offline version. This is due to the fact that we must initially set an estimate for optimal costs for the offline MTAC-E while the online version can use a random estimate for optimal costs and iterate towards a local minima that is more efficient. In addition to this, a higher estimate of optimal costs needs to be given for the offline version in order for faster execution time otherwise the completion time of the program could be significantly long. However, we note there is an increase in the sum of agent costs since in the online MTAC-E algorithm, each time an agent is considered for a new task, the cost is added to the total trajectory cost for that agent.

Run time comparisons are done on a laptop with a 2.6 GHz Intel i7-4720HQ processor using the time module in Python. Each run is done in simulation and execution times are measured against complete satisfaction of the given LTL specification. We see from Figure

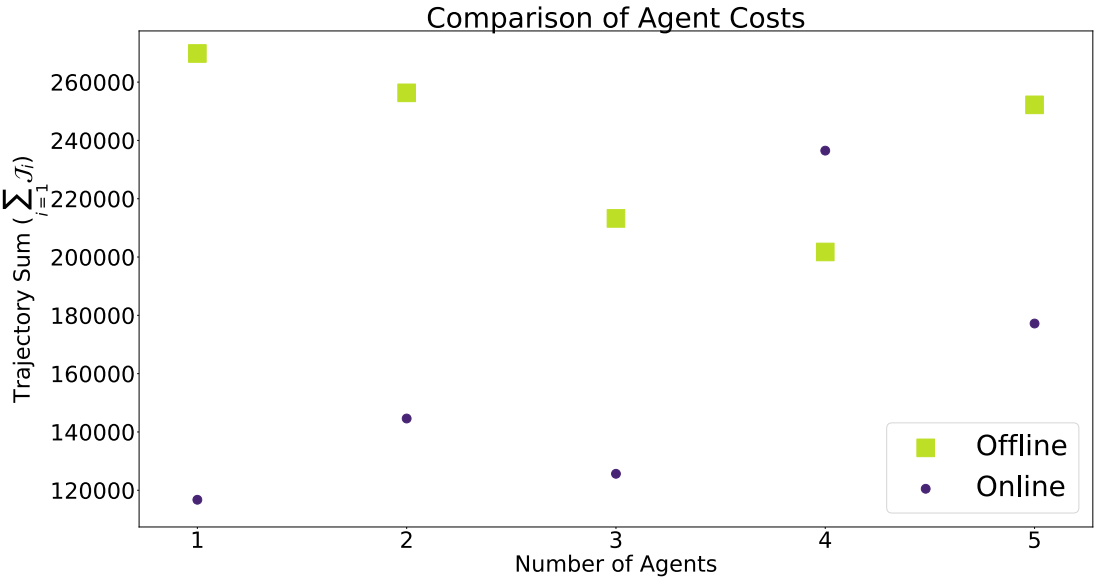


Figure 5.3: We compare the total trajectory costs for separate runs of the online and offline MTAC-E algorithm. We can see that the total costs associated with different numbers of agents is reduced for agents in the online case. This is due to a measurement of a known quantile function of a distribution and an optimal trajectory cost that is randomly initiated and updated at each sample step. This reduces the need of an expert to determine optimal costs for a particular domain.

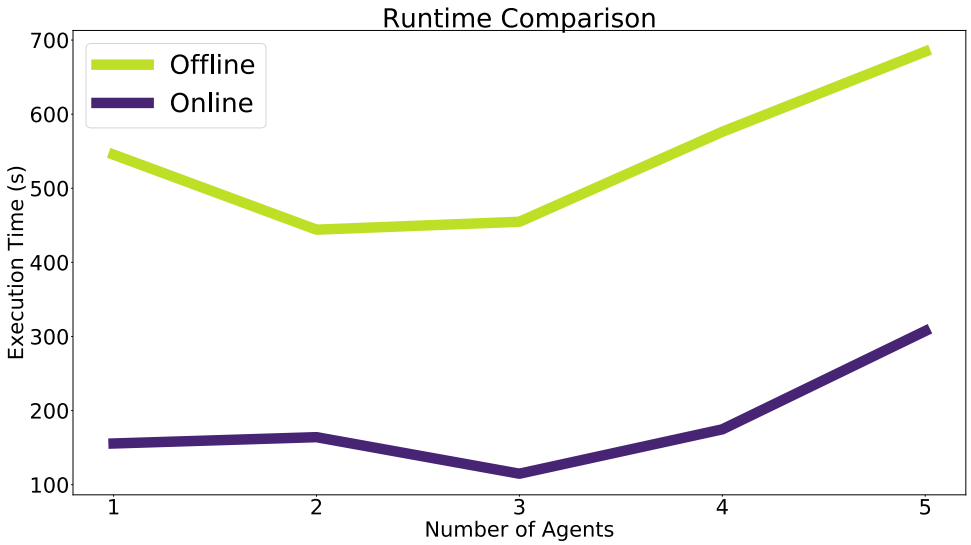


Figure 5.4: We compare the time to completion for both the online and offline MTAC-E algorithm. For all runs of the online MTAC-E with various numbers of agents considered, a lower total run-time is achieved.

5.4 a 3x - 5x factor of reduction of runtime for agents 1-5 indicating a faster algorithm for multi-agent task allocation.

From simulation results, we see the online MTAC-E algorithm is not only more efficient at minimizing the total costs for a multi-agent system but also in reducing individual agent costs and the overall run time associated with allocating tasks to a multi-agent system. We also verify our online algorithm experimentally in Section 5.6.

5.6 Fire Fighting Drones Experimental Results

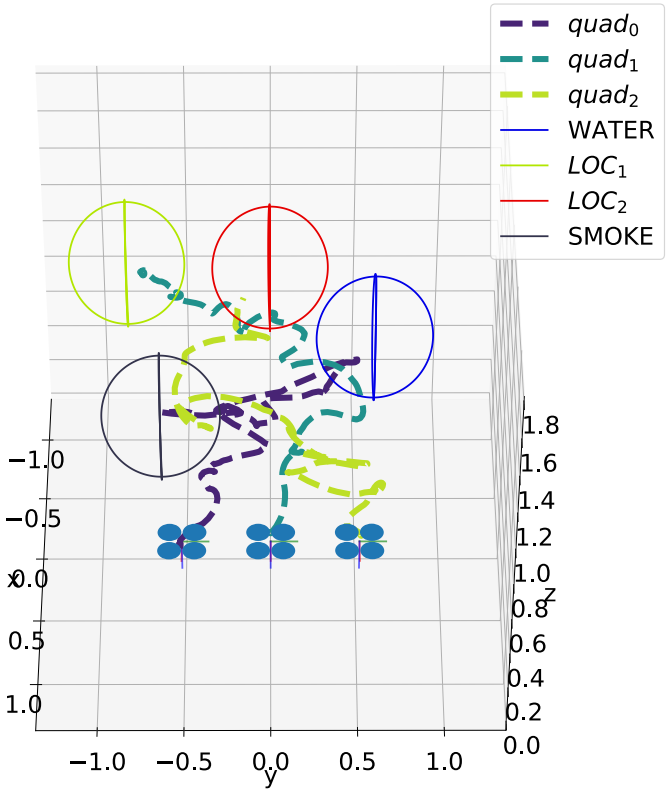
Experiments are validated on the Robotarium at Georgia Tech [113] using the Crazyflie 2.1 quadrotors. Control inputs are calculated from continuously differentiable splines generated online from desired waypoints using the differential flatness property of quadrotors described in Section 3.1.1. Quadrotor positions are tracked with a Vicon camera system with a tracking frequency of 100 Hz and the controller generates control inputs at a frequency of 50 Hz.

We recreate the simulated scenario of fire-fighting quadrotors by indicating desired regions of interest with hoops, characterized by ellipsoids, pictured in Figure 5.5b [1]. These hoops are covered with Vicon tracking markers, allowing us to record the corresponding center of the hoops and from the centers form the proposition sets. The regions of interest are satisfied if a quadrotor flies within 0.2 meters of the hoop. We give initial starting positions of

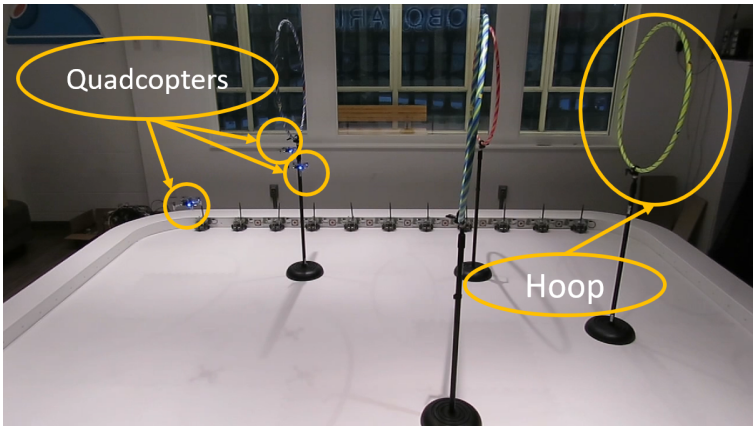
$$p_1 = \begin{bmatrix} 1 \\ -0.5 \\ 0.8 \end{bmatrix}^T, p_2 = \begin{bmatrix} 1 \\ 0 \\ 0.8 \end{bmatrix}^T, p_3 = \begin{bmatrix} 1 \\ 0.5 \\ 0.8 \end{bmatrix}^T$$

for three quadrotors and the desired global LTL specification $\phi = \Diamond LOC_1 \wedge \Diamond LOC_2 \wedge$

[1] Experiment Video: Online Multi-Agent Task Allocation via Cross Entropy



(a) The real trajectories of each quadrotor superimposed over the simulation environment. The locations of the hoops are the same as those in the experimental run.



(b) The quadrotors are flown in the Robotarium where hoops are used to validate the experiment. Each hoop has tracking markers to locate the center of mass of each hoop, used to calculate control points and the desired ellipsoids used to characterize the hoops. Quadrotors are also tracked with the Vicon system.

Figure 5.5: The trajectories of the quadrotors and a visualization of the experimental run are shown in the figure above.

$\square(SMOKE \implies CARRYING)$. We plot the real trajectories of the quadrotors in Figure 5.5a and given the initial positions, control inputs and global LTL specification we utilize the online MTAC-E algorithm to generate trajectories in real-time for each quadrotor. The assignments shown in the figure are dynamically allocated tasks given to each robot based on the individual cost function associated with each agent and environment constraints. These assignments are not strictly assigned and could change given new information (e.g. more samples from the online cross entropy algorithm) or a different cost function. We run the algorithm on a desktop with an Intel Core i7-7700K processor with 16 GB of RAM. The total run time of the experiment is 127 seconds and terminates when the entire desired sequence of hoops is satisfied for a specified LTL specification. Through this experiment, we show the implementation of the online MTAC-E algorithm on quadrotors which validates our task orchestration framework by decomposing a specification, delegating tasks and generating trajectories in **real-time** for a set of quadrotors.

5.7 Conclusion

In conclusion, we extend the multi-agent task allocation methodology to generate desired trajectories online via hyperellipsoid sampling and estimation of minimal quantile distribution differences. This allowed us to create a faster and more efficient method of trajectory sampling for multi-agent task allocation given an LTL specification that contained system and environmental constraints. To the authors knowledge, this is one of the first papers to develop online task allocation for quadrotors using cross entropy optimization and validate the algorithm experimentally on hardware. In addition to this we propose a formulation for the update step using the quantile functions of multivariate Gaussian distributions. We verify this in experiment and simulation as a task orchestration framework for decomposing and delegating tasks and generating trajectories for a multi-agent system to satisfy high-level user specifications given these constraints.

CHAPTER 6

LEARNING LTL FORMULAS USING SUPPORT VECTOR MACHINES

Robotic systems often need repeated sets of instructions to complete a task or set of tasks. These instructions can be trajectories, waypoints or more complex modalities [3, 114]. Sometimes, these tasks need to be repeated or are complex and may benefit from instructions specified in a syntax prevalent in system specification. Temporal Logics are well-known languages used for guaranteeing completion and persistence of system tasks [115]. In addition to this, the syntax is human-readable and easily interpretable for individuals familiar with logic syntax. Because of these benefits, many have used temporal logic for system specification design.

The goal of work centered around using temporal logic for robotic system control is to provide an intuitive and syntactically safe control modality for users to interact with robots. However, not all users are familiar with temporal logics and their syntax. The requirement to learn this language in addition to generating the correct formulas may be too large of a hurdle for some users to adopt. Thus, we propose to learn LTL specifications through desired traces.

Prior work on learning LTL specifications has focused on the use of learning a reward policy in a Markov decision process [9, 116]. However, this reward function shaping can become complicated and it is not always clear how to develop correct reward functions for a desired behavior. Other works focused on mining specifications given a desired set of traces and a template like the authors in [35]. This work accepts as input a log of traces and a general template [101] of an LTL specification. The authors develop a program that uses a tree-like search method to define LTL specifications with the given template that satisfy all traces given. However, this work required users to generate templates as well as a log of traces in order to develop the LTL formula.

Our work is similar to that of [10] and [39] in that we utilize templates for learning LTL specifications via demonstrations or given traces. In both works, the authors use Bayesian inference to infer the likelihood that a trace belongs to a chosen specification of a group of defined templates. We would like our model to learn over fewer trace examples and we refrain from probabilistic sampling techniques. In addition, the prior method may be computationally expensive for large datasets and relies on model assumptions implicit within its likelihood function, all of which we will avoid using support vector machines (SVMs).

In this chapter, we leverage support vector machines to learn desired LTL templates from a set of user generated traces. SVMs form a set of hyper-planes over a multi-dimensional (infinite dimensional) feature space to minimize a convex optimization problem [117]. SVMs are sufficient for classifying data from sparse support vectors which we consider ideal from the perspective that end-users may not have the capacity to generate thousands of features necessary for other learning methods. A key problem for correctly applying SVMs to a problem domain is identifying a linearly separable feature space that can be abstracted to a higher dimensional space. We propose to utilize a feature space composed of finite trace specific information. The trace features are generated from the trace length, a count on the propositions and we check for all LTL templates their equivalent non-deterministic finite automaton (NFA). Each NFA accepts a particular language and has an equivalent regular expressions, we check if each trace is accepted by these regular expressions.

We then propose a reduced LTL specification formulation to reduce the number of templates needed to define system behavior over a set of traces. We formulate this problem as a set covering problem which utilizes the SVM generated templates as sets. We demonstrate the efficacy of our work by comparing our learning algorithm to two brute force LTL mining techniques and we show simulation results with respect to a house surveillance robot case study. To the best of our knowledge, this is the first support vector machine learning method developed for learning temporal logic specifications from user generated traces.

This chapter proceeds in the following manner. We introduce LTL in Section 6.1. We develop our problem formulation in Section 6.2. The SVM learning model and feature space is presented in Section 6.3. We construct the reduced LTL specification in Section 6.4. Finally, the experimental results are provided in Section 6.5 and we provide concluding remarks in Section 6.6.

6.1 Preliminaries

In this section, we provide a brief background on LTL, a propositional logic well-suited for formally representing planning problems [17], system properties[60] and interpreting finite or infinite sequences [118, 115].

6.1.1 Linear Temporal Logic Definition

LTL specifications ϕ are defined as logic formalisms suited for specifying linear time properties[82, 23]. LTL specifications are defined over traces and indicate satisfaction of a set of propositions Π defined in the following definition.

Definition 12. *(Propositions) Let $\Pi = \{\pi_0, \ldots, \pi_k\}$ be the finite set of atomic propositions. Each proposition π_i maps from system state to true $(\top)$ or false $(\bot)$ and enables us to define a Boolean property of the state space (e.g. "Is the robot in area G?").*

LTL formulas over the set Π of atomic propositions are formed with the following logic operators:

$$\phi := \top \mid \pi \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \bigcirc \phi \mid \phi_1 \mathbf{U} \phi_2 \tag{6.1}$$

LTL formulas are composed of the basic Boolean operators (e.g. conjunction ($\wedge$) and negation ($\neg$)) and two temporal operators next ($\bigcirc$) and until ($\mathbf{U}$). Formula $\bigcirc\phi$ evaluates as true at time t if ϕ holds at time step $t+1$. Formulas of the form $\phi_1 \mathbf{U} \phi_2$ evaluates as true at time t if ϕ_1 is true until a time step is reached where ϕ_2 becomes true. From these base operators of LTL we can construct the following higher-order temporal operators:

$\Diamond$(eventually), $\Box$(always), and $\mathbb{W}$(weak until). The eventually formula ($\Diamond\phi$) holds true at time t if ϕ holds for some time $t_n \geq t$. The always formula ($\Box\phi$) holds true at t if ϕ holds at all times $t_n \geq t$. The weakly until formula ($\phi_1 \mathbb{W} \phi_2$) holds such that the occurrence of formula ϕ_2 does not need to be satisfied.

Definition 13. *(LTL Templates) LTL templates are a subset of LTL specifications with a particular structure that describe the relationship between trace events defined over an abstract set of atomic propositions* $\hat{\Pi}$. *From [101], LTL templates describe an essential aspect of system behavior (and the relationship between its propositions) that are expressed as an LTL specification.*

For example, the template $\Diamond A \implies \neg A \,\mathbf{U}\, (B \wedge \neg A)$ indicates "B must occur before A" which is the "precedence" relationship between atomic propositions A and B.

Definition 14. *(LTL Template Instance) Similar to [35], let* ϕ *be an LTL template. An LTL template instance* ϕ_i *of template* ϕ *has the same structure as template* ϕ *but the propositions are drawn from the trace demonstrations provided by a user. Let* $t : \hat{\Pi} \rightarrow \Pi$ *be a bijective assignment function mapping Boolean propositions from* $\hat{\Pi}$ *to the event propositions,* Π*, drawn from the user traces. The set of instances for LTL template* ϕ *is represented as* $\{\phi_i(t) : i \in \mathscr{I}\}$ *generated from the proposition set* Π *and template set* $\mathscr{I}$.

In the next section, we will introduce finite LTL, a subclass of LTL defined over finite traces.

6.1.2 Finite LTL

This chapter focuses on learning over user defined traces represented as finite sequences. Therefore, we consider finite LTL specifications. Finite LTL specifications are a class of LTL specifications designed to handle finite traces. One particular subclass of finite LTL specifications are known to be insensitive to infiniteness [119]. These specifications maintain the same properties as LTL defined over infinite traces except traces of propositions

will have the form

$$\pi = \pi_{finite}\{end\}^{\omega} \tag{6.2}$$

for some finite trace (π_{finite}) and a new proposition *end* which repeats infinitely (indicated by ω). These traces are interpreted as infinite traces where $\pi_{finite} \models \phi \quad iff \quad \pi_{finite}\{end\}^{\omega} \models \phi$. Thus, by verifying if a finite LTL specification is insensitive to infiniteness we can reduce it to an LTL specification that handles infinite traces of the form in Equation 6.2 where the finite traces satisfy the LTL specification if and only if its infinite counterpart does as well. All specifications referenced in this chapter are finite LTL specifications which are insensitive to infiniteness and will be referenced as LTL.

6.1.3 Automaton Construction

Any LTL specification can be represented via a non-deterministic finite automaton [34], which we define below.

Definition 15. *A non-deterministic finite automaton (NFA) is given as the tuple* $\mathscr{F} = (Q, Q_0, \beta, \delta, F)$ *such that:*

- Q *is a set of states*
- Q_0 *is a set of initial states*
- β *is the set of Boolean formulas defined over the proposition set (*Π*)*
- δ *is a set of transition conditions such that* $\delta : Q \times Q \rightarrow \beta$
- F *is a set of accepting final states.*

We utilize an equivalent deterministic interpretation for a particular type of NFA called a Deterministic Büchi Automaton (DBA) which allow infinite runs $q = q_0 q_1, \ldots$ where $q_i \in Q$. In addition, associated with each run is a sequence π – defined as a sequence

of propositions π_i from Π – which satisfies ϕ if it enables a run q such that $q_i \in F$ for infinitely many indices. This sequence σ enables transitions from state q to q'. These transitions, generated from $\delta(q,q') = \{\beta_i\}$, map onto a subset of the Boolean formulas, β, which evaluate to true if the proposition, π_i, from σ satisfies it. Moreover, the DBA can be constructed from an LTL formula ϕ where a finite sequence $\sigma_{finite} \models \phi$ if and only if $\sigma_{finite}\{end\}^{\omega}$ successfully produces a run q such that $q_i \in F$. In the next section, we will introduce our problem formulation.

6.2 Problem Formulation

We define our problem in two parts:

(a) First, consider the input $\hat{\pi}_i$ as a task demonstration represented as an observed sequence of propositions presented to an agent. Find a set of features, m, from trace $\hat{\pi}_i$ such that the output of our learning algorithm, $f(\hat{\pi}_i) \in \{\phi_i : i \in \hat{\mathscr{I}}\}$, matches the input trace to a template instance from Table 6.1 and $\hat{\mathscr{I}} = \mathscr{I}_1 \cup \cdots \cup \mathscr{I}_n$ is the union of template sets for n template types.

(b) Second, from a set of features – $\mathbf{X}_{N\times m}$ – for N traces, generate the LTL specification that satisfies all user provided traces as a composition of LTL templates. Then, develop a reduced LTL specification to limit the number of LTL templates needed to satisfy the user provided traces.

6.3 LTL Template Matching on SVMs

We propose that there exists a feature mapping between traces and a set of LTL templates. In addition to this, our goal is to show that there exists a linearly separable feature space that we use to classify traces to desired LTL templates. In this section, we give a brief overview of support vector machines (SVMs). We present the feature space of the trace set and detail how our multi-class SVM is trained for K templates. Lastly, we construct the

closed form expression of the full candidate LTL specification.

Table 6.1: Examples of LTL templates for general propositions π_i and π_j.

Patterns		LTL Templates
Occurrence	$\phi_{absence}(\pi_i)$	$\neg\Diamond(\pi_i \wedge \Diamond\pi_i)$
	$\phi_{existence}(\pi_i)$	$\Diamond\pi_i$
Order	$\phi_{response}(\pi_i, \pi_j)$	$\Box(\Diamond\pi_i \implies \pi_j)$
	$\phi_{precedence}(\pi_j, \pi_i)$	$\Diamond\pi_i \implies (\neg\pi_i \mathbf{U}(\pi_j \wedge \neg\pi_i))$
Choice	$\phi_{exclusive\ choice(\pi_i,\pi_j)}$	$(\Diamond\pi_i \vee \Diamond\pi_j) \wedge \neg(\Diamond\pi_i \wedge \Diamond\pi_j)$

6.3.1 Support Vector Machines

A support vector machine constructs a set of hyper-planes in a high or infinite dimensional space [117]. They are used for classification by finding the maximum separation between training points of any class and the margin boundaries. For training vectors $x_i \in \mathbb{R}^q, i = 1,\ldots,N$ and, in the two-class case, classes $y \in \{-1,1\}^n$, the goal is to find $w \in \mathbb{R}^q$ and $b \in \mathbb{R}$ such that the prediction class of $sign(w^T(K(x)+b)$ is correct for vectors x_i. This technique solves the convex optimization problem:

$$\min_{w,b,\xi} C\sum_{n=1}^{N}\xi_n + \frac{1}{2}||\mathbf{w}||^2 \tag{6.3}$$

$$s.t.\ y_i(w^T K(x_i)+b) \geq 1-\xi_i, \tag{6.4}$$

$$\xi_i \geq 0, i = 1,\ldots,n \tag{6.5}$$

where ξ_n is a slack variable introduced to softly penalize incorrectly classified points, $K(x_i)$ is the feature space representation of vector x_i and C controls the trade-off between the margin and slack variable penalty. In our case, we utilize multi-class SVMs and solve $\binom{K}{2}$ different 2-class SVMs for K classes where each class $k \in K$ represents an LTL template. In the following section, we describe our feature space in detail.

6.3.2 Trace Feature Space

From the input, a set of traces $\hat{\pi}$ provided by the user, we generate the feature space needed to map traces to desired LTL templates. Each trace $\hat{\pi}_i$ is composed of a sequence of propositions the user wants the robot to satisfy. These propositions are evaluated to true ($\top$) if the robot is located inside of the proposition at time t. For example, the trace $\hat{\pi}_i = [A, B, B, A]$ indicates the robot must satisfy proposition A, then B twice, followed by A again. We formulate the feature space as a composition of trace features generated from desired propositions we want a robot operating in the state space of our problem domain to satisfy.

Trace Features

- **Trace Length:** Each input trace $\hat{\pi}_i$ has a length associated with it. For example, the trace $\hat{\pi}_i = [A, B, B, A]$ has length 4. Each trace length is logged and added to the feature vector. The first feature is represented as

$$X_{i1}(\hat{\pi}_i) = \left[|\hat{\pi}| \right].$$

- **Regular Expression:** LTL is a logic formalism that can be represented by an equivalent NFA. A well-known property of NFAs is that the set of finite words accepted by some non-deterministic finite automaton A is its accepted language, denoted as $\mathscr{L}(A)$ [23]. In addition to this, there exists an equivalent regular expression for any non-deterministic finite automaton A [120] that accepts $\mathscr{L}(A)$. Thus, the set of traces accepted by this NFA is equivalent to the set of traces accepted by its regular expression. For every proposition, and combination of propositions, we generate equivalent regular expressions for the NFAs derived from the LTL templates. For example, the NFA of LTL specification $\phi_A = \Diamond A$ can be represented by the regular expression $\text{regex}(\phi_A) = (!A^*)(A^*)(1)^*$. Each of these regular expressions is added as a feature for the trace $\hat{\pi}_i$. If the entire trace is accepted in the language of the regular expression

we return its evaluation as 1 otherwise the regular expression returns 0. We express this relationship as $\text{regex}(\phi_{T_i}) : \hat{\pi}_i \to \{0,1\}$ for a regular expression on LTL template (T_i). We represent the second part of the feature vector as

$$X_{i2}(\hat{\pi}_i) = \begin{bmatrix} \text{regex}(\phi_{T_i[\hat{\pi}_{i:i\in|\Pi|}]}) \\ \vdots \\ \text{regex}(\phi_{T_k[\hat{\pi}_{i:i\in|\Pi|}]}) \end{bmatrix}^T . \tag{6.6}$$

- **Proposition Count:** Every trace is composed of propositions from the proposition set Π. We create a feature vector of size $2|\Pi|$ and count the number times each proposition appears in the trace. This feature vector follows the others with form

$$X_{i3}(\hat{\pi}_i) = \begin{bmatrix} count(\pi_i) \cdots count(\pi_{2|\Pi|}) \end{bmatrix} \tag{6.7}$$

From the trace features (X_{i1}, X_{i2}, and X_{i3}), we create the feature vector $X_i(\hat{\pi}_i)$ for each trace $\hat{\pi}_i$ of form:

$$X_i(\hat{\pi}_i) = \begin{bmatrix} \underbrace{X_{i1}(\hat{\pi}_i)}_{\text{trace length}} & \underbrace{X_{i2}(\hat{\pi}_i)}_{\text{regex features}} & \underbrace{X_{i3}(\hat{\pi}_i)}_{\text{prop. count features}} \end{bmatrix}^T \tag{6.8}$$

6.3.3 Training

Table 6.2: We use the HalvingGridSearch in scikit-learn to find the best parameters to train and test our SVM model. The following table shows the best parameters for two of each kernel tested.

data size	kernel fcn.	gamma	degree	C	mean acc.	sd.
1800	linear	–	–	10	90.8 %	0.079
5400	**linear**	–	–	**1**	**91.2** %	**0.076**
1800	rbf.	0.01	–	100	90.4 %	0.079
1800	rbf.	0.001	–	200	90.8 %	0.079
1800	poly.	–	2	100	90.8 %	0.079
1800	poly.	–	3	10	90.7 %	0.081

We train our multi-class SVM for all K LTL templates over $P_k(|\Pi|, p_i)$ instances of the k^{th} template where p_i is the number of propositions found in template k and $P_k(N, p_i)$ is the number of permutations for N items chosen p_i times. For instance, the eventually template $\phi_{eventually}$ for $|\Pi| = 3$ will generate three template types. For each of these template types, we generate an equivalent deterministic Büchi automaton (DBA) and sample 200 traces via random walks. After each trace is sampled, we find the associated feature vectors. We then train a one-vs-one multi-class SVM for various kernel functions (linear, radial basis function (rbf), polynomial) and show the results in Table 6.2.

6.3.4 LTL Template Classes

The returned classes of the SVM model will be the LTL templates as instances of all propositions in the proposition set Π. For LTL template ϕ_k, there are potentially p_k propositions selected by it where the total set of propositions is $\pi_{\mathbf{k}} \in \Pi^{p_k}$. The resulting LTL formula formed from the classification of new sample traces will then be selected from the set $\{\pi_{\mathbf{k}}\} \subseteq \Pi^{p_k}$. The full candidate LTL specification has the following form:

$$\phi = \bigwedge_{\pi_{\mathbf{k}} \in \{\pi_{\mathbf{k}}\}} \phi_k(\pi_{\mathbf{k}}) \tag{6.9}$$

Where each template ϕ_k potentially has a different number of propositions $\pi_{\mathbf{k}}$. For instance, if we consider the alphabet $\Pi = [A, B, C]$. The template $\phi_{eventually}$ will contain $p_k = 1$ propositions and a potential LTL candidate specification composed entirely of $\phi_{eventually}$ would be

$$\phi = \phi_{eventually}(A) \wedge \phi_{eventually}(B) \wedge \phi_{eventually}(C)$$

However, for K templates, the size of the candidate specification can grow exponentially in $\pi_{\mathbf{k}}$. This may result in very large end template specifications that over satisfy the given

traces. In addition, for brute force mining techniques [35, 121], this can be prohibitively expensive. We show in the next section how to formally represent our trace classification problem as a set cover problem where we minimize the number of templates needed to satisfy given traces.

6.4 Reduced LTL

Given that we have defined a way to match generated trace features to LTL templates using SVMs, we now consider the issue of over classification. By this we approach the problem of template matching for traces in the event that more LTL templates are selected than necessary. As we have mentioned in Section 6.3.4 the template space is exponential in the number of propositions. We would like to define a *minimal* specification that uses the smallest number of LTL templates necessary to satisfy all traces. We define minimal LTL specifications as the following:

Definition 16. *A minimal LTL specification is a composition of the smallest number of LTL templates that satisfies a set of given traces.*

In prior works [121, 9], minimal LTL specifications have been proposed. These works typically include solution techniques involving incrementally solving a Boolean satisfiability problem. We approach this problem as a set covering problem where we wish to minimize some objective function over the set of templates $\{\phi_i : \phi_i \in \phi_K\}$ for K templates. Formally, this problem is defined as:

$$\min \sum_{k \in K} w_k \phi_k \tag{6.10}$$

$$s.t. \sum_{k:\hat{\pi}_i \in k} \phi_k \geq 1 \qquad \forall \hat{\pi}_i \in \hat{\pi} \tag{6.11}$$

$$\phi_k \in \{0,1\} \qquad \forall k \in K \tag{6.12}$$

where $w_k \geq 0$ is the weight assigned to template ϕ_k, ϕ_k indicates if a set belongs in the set

cover and $\hat{\pi}_i \in k$ represents if the trace $\hat{\pi}_i$ belongs to the set covered by the template ϕ_k.

6.4.1 Minimal Set Identification

Recall that regular languages ($\mathscr{L}_1$) and ($\mathscr{L}_2$) are closed under union ($\cup$), meaning that $\mathscr{L}_1$ and $\mathscr{L}_2$ are regular languages and $\mathscr{L}_3 = \mathscr{L}_1 \cup \mathscr{L}_2$ is also a regular language [120].

In addition to this, the language accepted by the NBA A_{ϕ_1} is $\mathscr{L}(A_{\phi_1})$ and $\mathscr{L}(A_{\phi_2})$ is the regular language accepted by A_{ϕ_2} for LTL specifications ϕ_1 and ϕ_2, respectively. Therefore, if we construct an NBA $A_{\phi_3} = A_{\phi_1} \times A_{\phi_2}$ then both languages of A_{ϕ_1} and A_{ϕ_2} are accepted by A_{ϕ_3} such that $\mathscr{L}(A_{\phi_3}) = \mathscr{L}(A_{\phi_1}) \cup \mathscr{L}(A_{\phi_2})$.

Lemma 6.4.1. *If a minimal LTL specification ($\phi_{minimal}$) exists, it satisfies all user given traces.*

Proof. Assume the minimal LTL specification exists and does not satisfy all user given traces. Let a minimal LTL specification have form:

$$\phi_{minimal} = \bigwedge \phi_k, k = 1, \ldots, n. \tag{6.13}$$

For a set of traces $\{\hat{\pi}_{tr}\}$ that $\phi_{minimal}$ does not satisfy, the following must be true:

$$\begin{aligned} \{\hat{\pi}_{tr}\} &\notin \mathscr{L}(A_{\phi_{minimal}}) \\ \{\hat{\pi}_{tr}\} &\notin \mathscr{L}(A_{\phi_1}) \\ &\vdots \\ \{\hat{\pi}_{tr}\} &\notin \mathscr{L}(A_{\phi_n}) \end{aligned}$$

Therefore, $\phi_{minimal}$ must be reconstructed such that $\mathscr{L}(A_{\phi_{minimal}}) = \mathscr{L}(A_{\phi_{minimal}}) \cup \mathscr{L}(A_{\phi_t})$ where ϕ_t is the template, or set of templates, that covers $\{\hat{\pi}_{tr}\}$ and A_{ϕ_t} is the NBA constructed from ϕ_t. However, by definition, $\phi_{minimal}$ covers every element in the user given set. Therefore, if $\phi_{minimal}$ exists, all traces provided by the user are found in $\mathscr{L}(\phi_{minimal})$

and a contradiction is reached. □

Remark 6.4.1. *This formulation relies on perfect trace classification to the set of LTL Templates. However, since our sets are generated from the SVM model with results presented in Table 6.2 there may be misclassifications in practice. Therefore, $\phi_{minimal}$ will in practice produce $\phi_{reduced}$ which generates set covers that cover all traces with a misclassification rate $\tilde{\pi}_i$ for each trace $(\hat{\pi}_i)$ where $\tilde{\pi}_i = \frac{b_K}{K}$ where b_K is the number of incorrect template member assignments for $\hat{\pi}_i$ and K is the number of templates. The total misclassification rate is then*

$$r(\pi^*) = \frac{\sum^N \tilde{\pi}_i}{|\hat{\pi}|}. \tag{6.14}$$

6.4.2 LTL Template Composition

We introduced scenarios for the explosion of template size for SVM classification in Section 6.3.4. Next we show that the minimal LTL specification is bounded in size.

Lemma 6.4.2. *The length of the minimal LTL specification* $(|\phi_{minimal}|) \leq \sum_{i:S_j \in f}^{j} |\phi_{S_j}| \cdot \sum_{k:k \in \phi_k^*} |\phi_k|$

Proof. We demonstrate this in four (4) steps:

1. We construct the dual linear program (LP) relaxation of our minimal LTL problem as:

$$\max \sum_{i=1}^{n} \hat{\pi}_i \tag{6.15}$$

$$s.t. \sum_{i:m_i \in S_j} \hat{\pi}_i \leq w_j, \qquad j = 1,\ldots,m \tag{6.16}$$

$$\hat{\pi}_i \geq 0, \qquad i = 1,\ldots,n \tag{6.17}$$

where m_i is a trace belonging to set S_j.

2. Any feasible solution of our dual LP $\leq$ feasible solution of our primal LP (the LP relaxation of the minimal LTL problem):

$$\sum_{i=1}^{n} \hat{\pi}_i \leq \sum_{k=1}^{m} w_k \cdot \phi_k \tag{6.18}$$

From the weak-duality theorem of linear programs [122], we have

$$\sum_{i=1}^{n} \hat{\pi}_i \leq L^*_{LP} \leq OPT \tag{6.19}$$

where L^*_{LP} is the optimal solution of the LP and OPT is the optimal solution for the minimal LTL problem.

3. If we consider a tight constraint for the dual LP (e.g. $\sum \hat{\pi}_i = w_j$) we define $k \in K'$ of the set cover K' with subsets S_j only if $w_j = \sum_{i:\hat{\pi}_i \in S_j} \hat{\pi}_i{}^*$, then we have the following f-approximation:

$$\sum_{k \in K'} w_k = \sum_{k \in K'} \sum_{i:\hat{\pi}_i \in S_j} \hat{\pi}_i{}^* \tag{6.20}$$

$$= \sum_{i=1}^{n} |k \in K' : \hat{\pi}_i \in S_j| \cdot \hat{\pi}_i{}^* \tag{6.21}$$

$$\leq \sum_{i=1}^{n} f_i \hat{\pi}_i{}^* \tag{6.22}$$

$$\leq f \sum_{i=1}^{n} \hat{\pi}_i{}^* \tag{6.23}$$

$$\leq f \cdot OPT \tag{6.24}$$

where $f_i = |k \in K' : \hat{\pi}_i \in S_j|$ is the size of the number of sets that contain $\hat{\pi}_i$ and $f = max_{i=1,\ldots,n} f_i$.

4. It is well known that the procedure above is used to develop a dual rounding algorithm to form an f-approximation algorithm of the set cover problem [122, 123,

124]. The solution to the primal LP will be no greater than $f \cdot OPT$ and the length of $\phi_{minimal}$ will thus be no greater than the length of templates in f chosen to cover any trace ($\hat{\pi}_i$) times the length of templates in the optimal solution ($|\phi_k|$).

□

In the next section, we will evaluate our model against brute force techniques and verify results over simulation through a house surveillance robot case study.

6.5 Experimental Results

In this section we will evaluate the performance of our classification technique compared to brute force techniques Samples2LTL [121] and Texada [35]. Next, we show how our technique can be used in a house surveillance scenario where a user gives a robot a set of traces and the robot learns the desired behavior from this set of traces.

For all our applications, we consider the minimal LTL specification set weights $\hat{w}_k$. The standard weights are represented as

$$\hat{w}_k = \sum_{i:\hat{\pi}_i \in \phi_k} \hat{\pi}_i \tag{6.25}$$

where each weight sums over the number of traces satisfied by template ϕ_k. We enhance these weights through the following equations

$$c_w = \frac{1}{(N+1) \otimes \mathbf{1}_k^T - c_{\text{bin}}^T} \tag{6.26}$$

$$w_k = \frac{\hat{w}_k ||c_w||}{c_w} \tag{6.27}$$

where c_{bin} represents the bin count where an array is indexed by the template numbers K and counts which traces were classified by the SVM model and updates the bin number accordingly and N is the number of traces. This new weight biases our set cover to prefer

templates that were chosen from the SVM formulation. The new weight w_k is calculated in Equation 6.27.

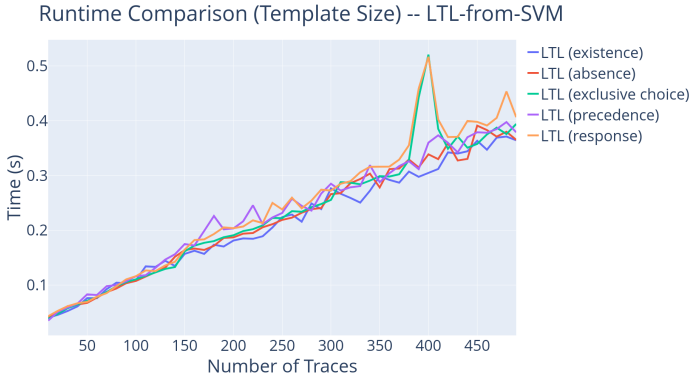
6.5.1 Comparison to Brute Force Techniques

We compare our LTLfromSVM algorithm against the brute force algorithms Texada and Samples2LTL. We generate trace batches ranging in size from 10-490 traces for the set of templates LTLfromSVM can identify. Runtime comparison is generated for each template for an increasing number of traces. Experiments are conducted on a 64-bit Ubuntu Machine with 16 GB RAM and an Intel i7 Skylake processor. In Figure 6.1b we show that Texada outperforms LTLfromSVM in runtime due to performance increases utilized in C++. However, we show LTLfromSVM maintains consistent performance over increasing template size. As desired template sizes grow in specification size, Texada requires additional time to find satisfactory specifications. In addition to this, template types must be provided to Texada an input that is not required for LTLfromSVM.

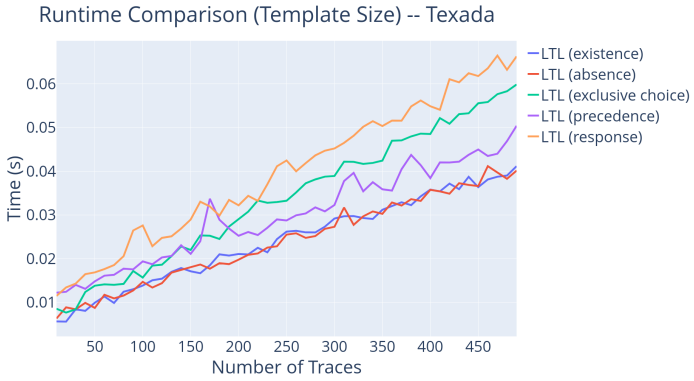
The runtime performance of brute force technique Samples2LTL, presented in [121], is shown in Figure 6.1c. Both LTLfromSVM and Samples2LTL are written in Python and we see LTLfromSVM performs 10x faster in trace classification compared to Sample2LTL. In addition to this, LTLfromSVM also maintains better performance consistency compared to Samples2LTL.

6.5.2 Case Study: House Surveillance Robot

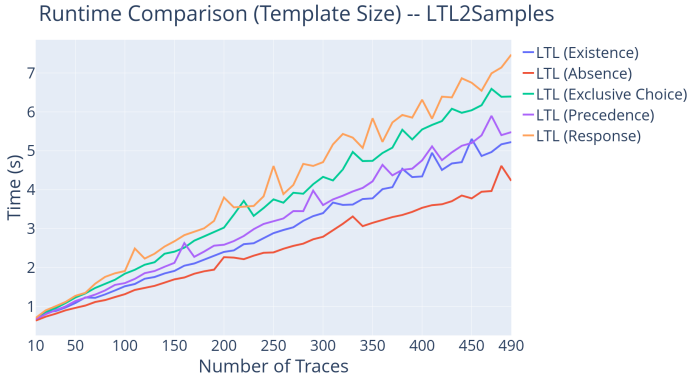
We present an application of the LTL SVM learning method via a house surveillance drone. The problem domain is a set of five locations (Kitchen, Bathroom, Bedroom, Living Room, and Laundry Room) that can be surveyed by a drone depicted in Figure 6.2. These locations are represented as propositions $\pi_k \in \Pi$. Additionally, these locations are represented as ellipsoids in $\mathbb{R}^3$. The goal of our model is to learn a satisfying LTL specification given a sequence of traces. We utilize the differential flatness property of quadrotors to generate



(a) Runtime performance of LTLfromSVM algorithm over the set of template types.

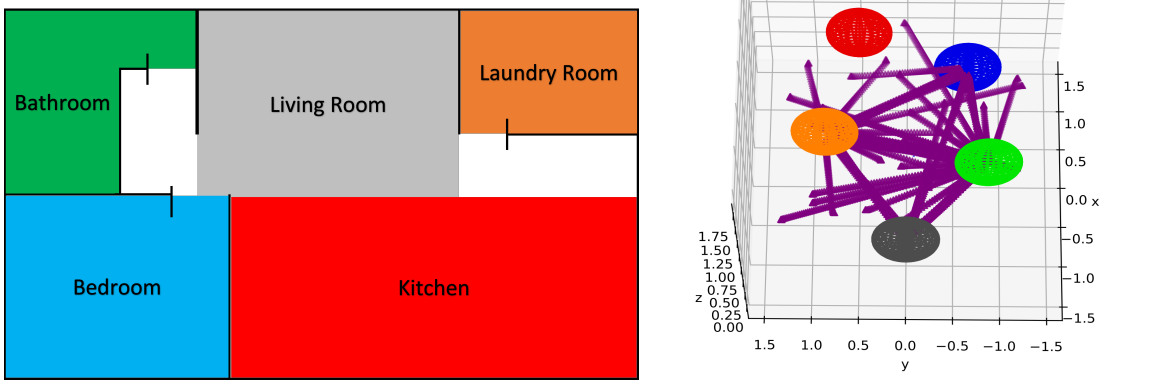


(b) Runtime performance of Texada over the set of template types accepted by LTLfromSVM.



(c) Runtime performance of Samples2LTL for increasing trace size over traces drawn from expected template types.

Figure 6.1: A runtime comparison of LTLfromSVM, Texada and Samples2LTL for each template type over an increasing number of traces.



(a) A 2D map of the problem domain. We present a case study of the LTL SVM learning algorithm on a house surveillance drone.

(b) We show the 30 trajectories generated from the user defined traces.

Figure 6.2: In Figure 6.2a we depict a map of the problem domain. The drone has access to five locations in a house. The user must give a set of traces (task demonstrations) which will be used to learn a LTL specification that satisfies this desired behavior. In Figure 6.2b, we show the desired trajectories corresponding to the input traces.

trajectories in the flat output space [106, 75]. We generate three-time continuously differentiable trajectories $\eta(t) \in C^3$ and control the simulated dynamical system via the inputs derived from the flat outputs. We present the results of our model via two examples.

In the first example, we generate 200 traces from an LTL specification that contains behaviors we would like our surveillance drone to adopt. We give the specification

$$\begin{aligned} \phi = \phi_{eventually}(\text{Living Room}) \wedge \\ \phi_{precedence}(\text{Laundry Room, Bathroom}) \end{aligned} \tag{6.28}$$

which means "eventually survey the living room and ensure the drone visits the laundry room before the bathroom". We train the SVM with a linear kernel over the 200 traces that

satisfy the expected specification and receive the following learned LTL specification:

$$\begin{aligned}\phi = \phi_{\textit{exclusive choice}}(\text{Bathroom, Laundry Room}) \wedge \\ \phi_{\textit{exclusive choice}}(\text{Bathroom, Living Room}) \wedge \\ \phi_{\textit{response}}(\text{Bathroom, Laundry Room}) \wedge \\ \phi_{\textit{precedence}}(\text{Laundry Room, Bathroom})\end{aligned} \tag{6.29}$$

While this specification does not exactly match the desired specification, the traces that satisfy this specification provide the desired behavior of "visiting the laundry room before the bathroom" and in some traces "visiting the living room". Also, the learning method does return the precedence template and does not necessarily violate the conditions of our "original" specification.

When we apply the minimal LTL formulation from Section 6.4, the reduced specification

$$\phi_{\textit{precedence}}(\text{Laundry Room, Bathroom})$$

is produced with a misclassification rate $r(\pi^*) = 38.4\%$. This rate indicates that of the 200 traces mapped to the template classes, 38% of the traces will be incorrectly assigned to a particular template class.

The second example is conducted under more practical assumptions. We hand generate 30 traces and simulate trajectories for quadrotors that satisfy the propositions within each trace (see Figure 6.2b). We generate the specification

$$\begin{aligned}\phi_{\textit{response}}(\text{Laundry Room, Bathroom}) \wedge \\ \phi_{\textit{response}}(\text{Living Room, Laundry Room}) \wedge \\ \phi_{\textit{absence}}(\text{Kitchen}).\end{aligned}$$

Note that the response template is present in this LTL specification. This is due to the high misclassification rate relative to other templates seen in Figure 6.3a where response templates are indexed from 16-35, however, we have correctly identified the $\phi_{absence}$ template as well as generating a template that satisfies the given traces.

The reduced specification generated is produced below

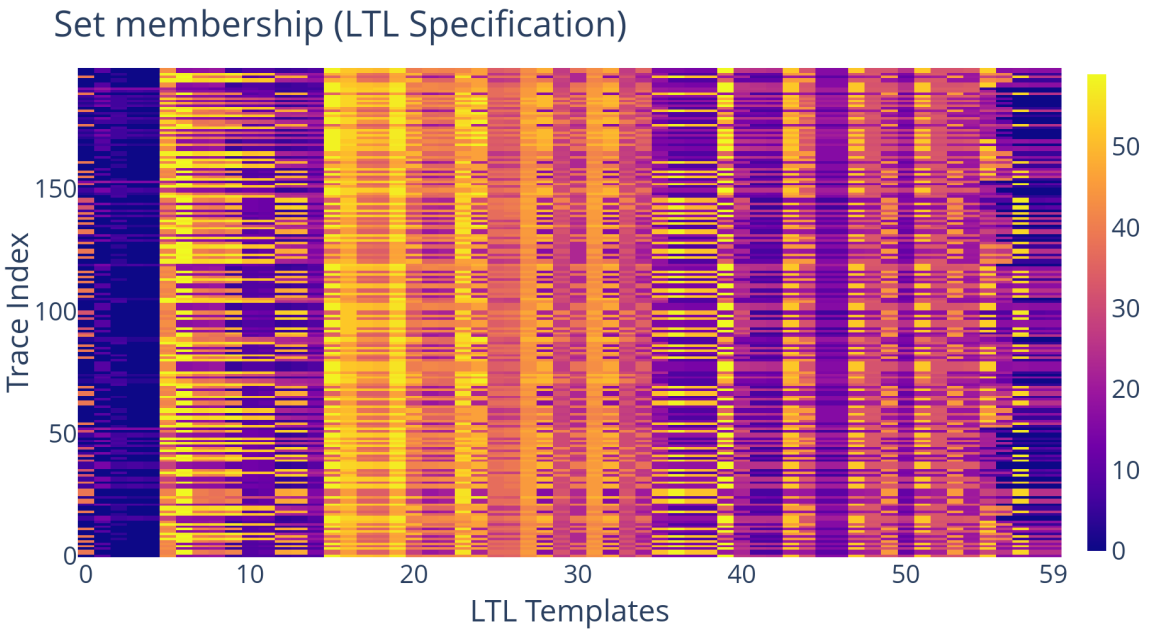
$$\phi_{response}(\text{Laundry Room, Bathroom})$$

with a misclassification rate $r(\pi^*) = 25.5\%$. It is likely that better feature sets and appropriately chosen kernels will fix the generalization issues present with the current framework. From Figure 6.3b, the templates belonging to the $\phi_{response}$ templates are labeled 16-35. The highest density of template assignment is issued to these templates.

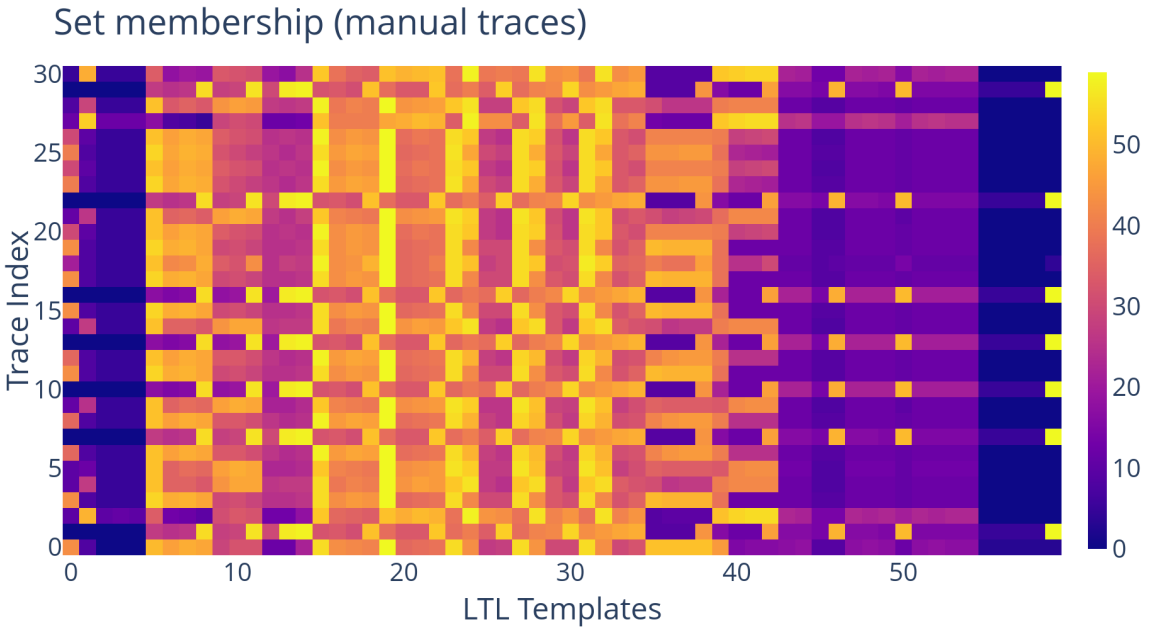
It is a known problem that learning template matching without a unified metric is hard [39]. In fact, by addressing this problem we seek to identify a metric space that appropriately defines the distance between a set of graphs given paths through the graph. This work shows preliminary steps towards addressing this problem from the perspective of LTL template matching.

6.6 Conclusion

In conclusion, we have developed a method of leveraging finite LTL specifications to generate a feature space used for SVM classification of traces to LTL templates. We show the accuracy of the best performing models using the feature set involving traces length, regular expressions and proposition counting over the set of traces. We formulate the reduced LTL specification problem as a set cover and show it produces the smallest length specification associated with a misclassification rate and a set of traces. In addition to this, we compare our algorithm to brute force LTL mining techniques and apply it to two example problems.



(a) This heatmap represents the set membership of 200 traces classified from the LTL specification in Equation 6.28.



(b) This heatmap represents the set membership of the 30 traces generated from a user.

Figure 6.3: A heatmap representing the number of times a particular trace (y-axis) was classified into the set defined by the LTL Templates (x-axis) learned via the SVM. The color gradient increases from low template membership (purple) to high template membership (yellow) for each trace.

CHAPTER 7

REMOTELY ACCESSIBLE AERIAL SWARMS

Aerial robots are increasingly becoming more prevalent in robotics research and industrial applications. They are often agile and lightweight and able to maneuver in spaces that ground vehicles are unable to reach [125]. As more research is done on these types of robots, researchers will need to have access to the resources necessary to accommodate aerial vehicles. Over time, access to aerial robotic platforms has grown with many types of vehicles accessible to users [126]. However, there still remains a barrier to entry for aerial swarm robotics researchers due to the space and hardware requirement and additional background knowledge necessary to instantiate an aerial robotics research testbed. Primarily, it is time consuming and expensive to develop a state-of-the-art testbed capable of facilitating multiple aerial robots for swarm robotics research. In addition to this, a remotely accessible testbed must consider how to ensure hardware safety while providing users the ability to control large groups of robots.

The Robotarium solved this problem for ground vehicles by being the first remote access multi-robot testbed [113, 127]. It allowed users to access a sandbox-like environment through an API where users can control dozens of robots for a wide variety of multi-robot experiments. In addition to this, safety constraints are applied to each users algorithms to ensure continuous autonomous operation of the Robotarium by limiting damage to the platform and individual robots. To date, thousands of experiments have been executed on the Robotarium and users ranging from individuals to research institutions have tested their algorithms on the testbed.

In this chapter, we introduce an extension of the Robotarium by providing remote access to aerial vehicles for multi-robot research. We do this by providing users access to micro-UAVs (Unmanned Aerial Vehicles) i.e. quadrotors, where users can test algorithms and

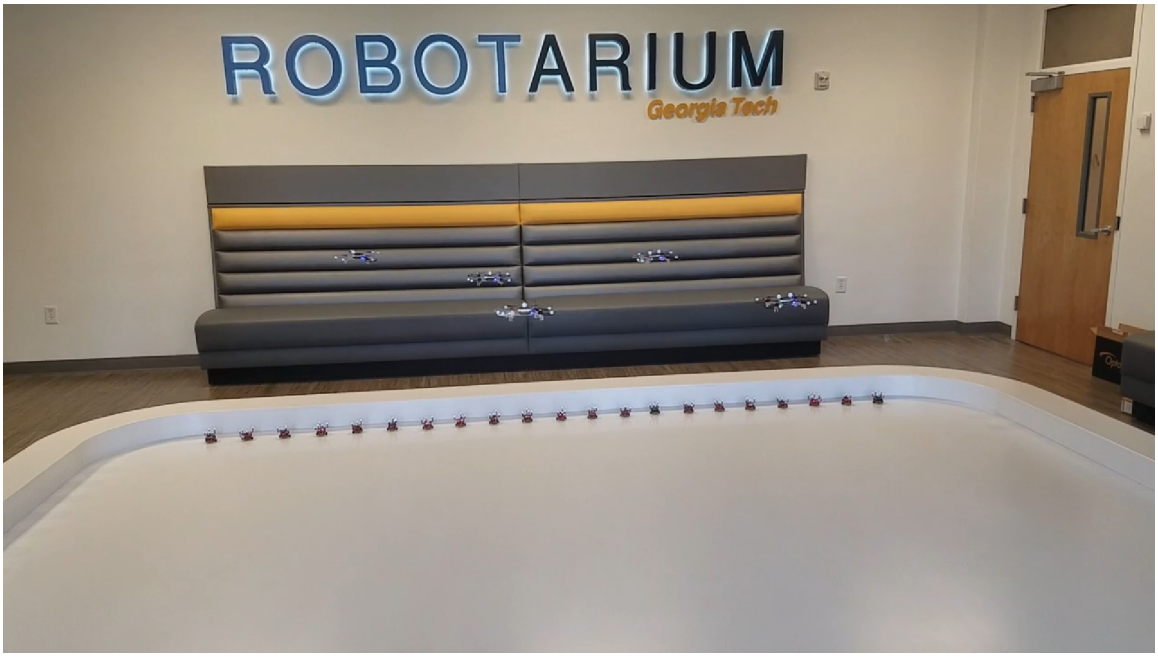


Figure 7.1: A quadrotor demonstrating the collision avoidance behaviors generated via exponential barrier functions. This experiment was conducted with 5 quadrotors on the Robotarium testbed.

controllers on this platform with minimal knowledge necessary. We provide users access to quadrotors through an API with encoded safety measures to ensure long duration autonomy and repeatable use.

Aerial robotics research platforms are increasingly being developed [128, 129]. The RAVEN testbed [130] was an early approach to multi-vehicle testbeds using motion-capture systems which included aerial vehicles. It studied long duration autonomy with UAVs and facilitated deployment of coordinated control algorithms between ground and air vehicles. OpenUAV is an open source cloud-based aerial swarm robotics platform that allows external users to run simulations using their code base [131]. These works differ from our testbed in that we provide open access to *simulations and physical hardware* through *safe* interactions to bridge the simulation-experiment gap commonly seen in multi-robot system research for external users.

The Robotarium provides users a simple interface to interact with multiple UAVs while guaranteeing safety through minimally invasive control input augmentations. A necessary consideration in the development of this addition to the Robotarium is what level of ac-

cess do we allow the novice user to have for quadrotors. These dynamical systems are highly nonlinear under-actuated vehicles that require extensive knowledge of their dynamics for control design. In order to democratize access to aerial vehicles while incorporating safe features in the design, we develop a system architecture composed of hardware, software and control-theoretic design to maintain long duration autonomy and satisfaction of user experiments. We leverage the Crazyflie quadrotor as the aerial vehicle platform for all experiments conducted on our testbed due to its small size and modular design which makes it well suited for indoor testbeds like ours. VICON tracking is used for maintaining global positioning of each quadrotor during complex flight maneuvers. Communication is done using ROS where we leverage the publisher-subscriber model of message passing. For communicating from a host PC to large aerial swarms, Crazyswarm [132], a system architecture for state estimation and communication of aerial swarms, is leveraged.

This chapter proceeds in the following manner. In Section 7.1 we introduce the quadrotor model and our formulation for providing users a method for controlling quadrotors. In Section 7.2 we discuss the hardware necessary for using quadrotors on the testbed. Section 7.3 will layout the safety features added for aerial vehicles including control barrier functions and firmware changes on-board the quadrotors. Section 7.5 details the simulator details and safety verification process. Section 7.6 experimentally validates the Robotarium as a testbed for developing and testing algorithms designed for aerial swarms. Finally, we provide concluding remarks in Section 7.7.

7.1 Leveraging Differential Flatness for Quadrotor Control

The dynamics of quadrotors are well-modelled systems that utilize the thrust generated by four propellers to move in three-dimensional space. They are complex dynamical systems that require a well-defined understanding of the dynamics and limitations of these systems. In this section, we give a brief overview of quadrotor dynamics and the waypoint controller we provide external users.

7.1.1 Quadrotor Model

The inertial frame and body-fixed frame are labeled $\{I\}$ and $\{B\}$ with right-handed coordinates, respectively. Left subscripts represent the reference frame with respect to which quantity is expressed. We will use the $Z-Y-X$ Euler angle convention to express the rotation matrix of the quadrotor system. Using this convention, the rotation matrix from $\{B\}$ to $\{I\}$ is ${}^{I}R_{B} = R_z(\psi)R_y(\theta)R_x(\phi)$ with ϕ, θ and ψ representing the roll, pitch and yaw angle, respectively and $R_z(\cdot), R_y(\cdot),$ and $R_x(\cdot)$ indicating the rotation about the body frame $z, y,$ and x axes.

The equations of motion describing the dynamics for a quadrotor are:

$$ {}_{I}\ddot{\boldsymbol{r}} = -{}_{I}\boldsymbol{e}_z g + \frac{1}{m}\left({}^{I}R_{B}\,{}_{B}\boldsymbol{e}_z f_t\right) \tag{7.1} $$

$$ {}_{B}\dot{\boldsymbol{\omega}} = {}_{B}\mathscr{I}^{-1}\left(-{}_{B}\boldsymbol{\omega} \times \mathscr{I}\,{}_{B}\boldsymbol{\omega} + {}_{B}\tau\right) \tag{7.2} $$

where the linear acceleration of the quadrotor is expressed in Equation 7.1 where ${}_{I}\ddot{\boldsymbol{r}} = [\ddot{x}, \ddot{y}, \ddot{z}]^T$ is the acceleration, g is the acceleration due to gravity, m is the mass of the vehicle, f_t is the total thrust generated by the rotors and ${}_{B}\boldsymbol{e}_z$ is the unit vector along the body frame z-axis.

The angular acceleration of the quadrotor, expressed in the body frame $\{B\}$, is represented in Equation 7.2 where ${}_{B}\dot{\boldsymbol{\omega}} = [p, q, r]^T$ is the angular velocity vector, ${}_{B}\tau = [\tau_x, \tau_y, \tau_z]^T$ is the torque vector and ${}_{B}\mathscr{I} \in \mathbb{R}^{3\times 3}$ is the moment of inertia matrix. We consider the inertia matrix to be a diagonal matrix (e.g. ${}_{B}\mathscr{I} = \text{diag}(\mathscr{I}_x, \mathscr{I}_y, \mathscr{I}_z)$) due to the assumed symmetry and choice of inertial axes. For controller design, we consider the system input $\boldsymbol{u} \in \mathbb{R}^4$ to be the total thrust and torques generated by each rotor.

7.1.2 Differential Flatness

Differential flatness is a well known property of many dynamical systems [74, 75]. This property allows the expression of the full state and input of the system as algebraic func-

tions of the flat output and its time derivatives.

It has been show that quadrotor dynamics are differentially flat in [5, 133], thus we can generate trajectories that leverage the nonlinear dynamics of the quadrotors rather than considering the system dynamics as constraints. We use the controller developed in [5] onboard the Crazyflie where the flat outputs for the quadrotor are $\boldsymbol{h} = [x, y, z, \phi]^T$. The full state $\boldsymbol{\xi} = [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, p, q, r]^T$ and input $\boldsymbol{u} = [f_t, \tau_x, \tau_y, \tau_z]^T$ of the system are represented algebraically using the following functions

$$\boldsymbol{\xi} = g_s(\boldsymbol{h}, \dot{\boldsymbol{h}}, \ddot{\boldsymbol{h}}, \dddot{\boldsymbol{h}})$$
$$\boldsymbol{u} = g_u(\boldsymbol{h}, \dot{\boldsymbol{h}}, \ddot{\boldsymbol{h}}, \dddot{\boldsymbol{h}}, \ddddot{\boldsymbol{h}})$$

for endogenous transformations (g_s, g_u).

7.1.3 Quadrotor Waypoint Control

We will leverage the differentially flat properties of quadrotors for trajectory tracking. We do this by allowing users to control the quadrotors using desired pose commands, $r_d = [x, y, z]^T$, and generating four-times differentiable desired trajectories (η_d) via spline interpolation.

We generate interpolating splines p_i for $n+1$ points $(s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n), n \geq 3$ where s is a pose dimension and t is the time. In our case we set a dimension of the current pose, r, to s_0 and the desired pose, r_d to s_n. Two additional midpoints are found to meet the requirements of the minimum number of interpolating points. The function $P(s)$ is a piecewise continuous function that must satisfy the following conditions:

1. For each interval $[s_{j-1}, s_j]$, $j \in \{1, 2, 3, \ldots, n\}$, $P(s)$ is given by the polynomial $p_j(s)$.

2. $P(s)$ satisfies the interpolation conditions: $P(s_j) = t_j$ for all $j \in \{0, 1, 2, \ldots, n\}$ (i.e. $p_j(s_{j-1}) = t_{j-1}$, $p_j(s_j) = t_j$ for all $j \in \{1, 2, 3, \ldots, n\}$).

3. $P(s)$ is four-times differentiable in $[s_0, s_n]$, i.e. for each $j \in \{1, 2, \ldots, n-1\}$ it holds that $p'_j(s_j) = p'_{j+1}(s_j)$, $p''_j(s_j) = p''_{j+1}(s_j)$, $p'''_j(s_j) = p'''_{j+1}(s_j)$ and $p''''_j(s_j) = p''''_{j+1}(s_j)$.

4. At the endpoints, the curvature of $P(s)$ is set to zero such that: $p''''(s_0) = p''''(s_n) = 0$.

Each 5^{th} degree polynomial, $p_j(s) = a_j t^5 + b_j t^4 + c_j t^3 + d_j^2 t + e_j t + f_j$, has six unknown coefficients. For n 5^{th} degree polynomials, $6n$ equations are needed to solve for the coefficients. These coefficients can be found by solving a system of equations, at each dimension, thus giving our desired trajectory $n_d \in C^4$ as a function of time (t).

From the spline interpolation, we generate a virtual control input $v \in \mathbb{R}^3$ to control integrator dynamics and we set the yaw angle to zero, $\psi(t) = 0$. Using this, we have

$$\dddot{\dddot{r}} = v \tag{7.3}$$

where $r = \eta_{1:3} = [x, y, z]^T \in \mathbb{R}^3$ and v is the virtual snap control input. The integrator system, written in state space form, is

$$\dot{q} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{F \in \mathbb{R}^{4\times 4}} \otimes I_{3\times 3} \cdot q + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{G \in \mathbb{R}^4} \otimes I_{3\times 3} \cdot v, \tag{7.4}$$

where $q = [r^T, \dot{r}^T, \ddot{r}^T, \dddot{r}^T]^T \in \mathbb{R}^{12}$ and $\otimes$ is the Kronecker product. The integrator system is driven to the virtual input v using standard linear feedback control techniques and the corresponding state values of the integrator model are provided to the *FullStateSetpoint* streaming point provided by Crazyswarm. Streaming points allow controllers to send setpoint values over radio at high frequencies compared to other methods of sending message packets to Crazyflies. This allows us to devote PC computation to high-level planning or other resource intensive tasks while only sending the control setpoints to the quadrotor.

Figure 7.2: Crazyflie 2.1 quadrotors are an open source, lightweight aerial vehicle platform well suited for research applications. Global positioning is acquired via a VICON optical tracking system that tracks the center of mass of uniquely positioned markers for each quadrotor.

This setpoint takes as input $(r^T, \dot{r}^T, \ddot{r}^T, \psi, p, q, r)$ which can be generated via integrator dynamics or differential flatness. This streaming setpoint is then received by the onboard feedback controller to achieve the desired states where, in our case, this controller is the one described in [5].

7.2 Hardware

In this section, we overview the hardware that makes the Robotarium a capable multi-quadrotor testbed. We leverage much of the platform developed for the Robotarium [127] except with a few key changes made explicitly for the introduction of quadrotors.

7.2.1 Quadrotor Tracking

The Robotarium operates on a 3.65 m $\times$ 4.25 m $\times$ 2 m arena. quadrotor experiments can be recorded via either an overhead camera above the testbed or a side camera. Position tracking is provided via a VICON motion-capture system. We use 16 VICON Vantage cameras and track unique tracking marker configurations for each quadrotor at a rate of

120 Hz, see Figure 7.2.

7.2.2 Crazyflie Robots

One main challenge of providing a remotely accessible testbed is to find an appropriate platform for running user experiments. The quadrotors we use must be easily repairable and durable in the event of collisions or damage. In addition to this, they must be small enough to use on our testbed and amenable to swarm robotics research. We use the Crazyflie 2.1 [1], an open source, lightweight quadrotor platform well suited for research in aerial vehicles. Each Crazyflie weights 27 grams with a battery, however, we include a VICON tracking hat and markers and Qi charging reciever which increases the weight to 39 grams. Currently, up to 8 quadrotors are available to the user for swarm robotics experiments where we rely on the Crazyswarm communication architecture for large scale communication from the host PC to Crazyflies through broadcasted messaging. All Crazyflies are controlled via the Crazyradio PA, a USB radio dongle designed for low latency communication. In addition to this, each Crazyflie is equipped with a 260 mAh battery, capable of approximately 4 minutes of flight time depending on the aggressiveness of user experiments submitted.

7.2.3 Wireless Charging

Autonomous wireless charging is essential for long duration testbed operation. In order to maintain automatic experiment execution for users without the additional overhead time needed for battery changing, we utilize wireless charging. We use Qi inductive charging decks from Bitcraze in addition to transmitting coils set within a 3D printed mounting dock. In Section 7.4, we will explain in detail the autonomous charging routine used for autonomous testbed operation.

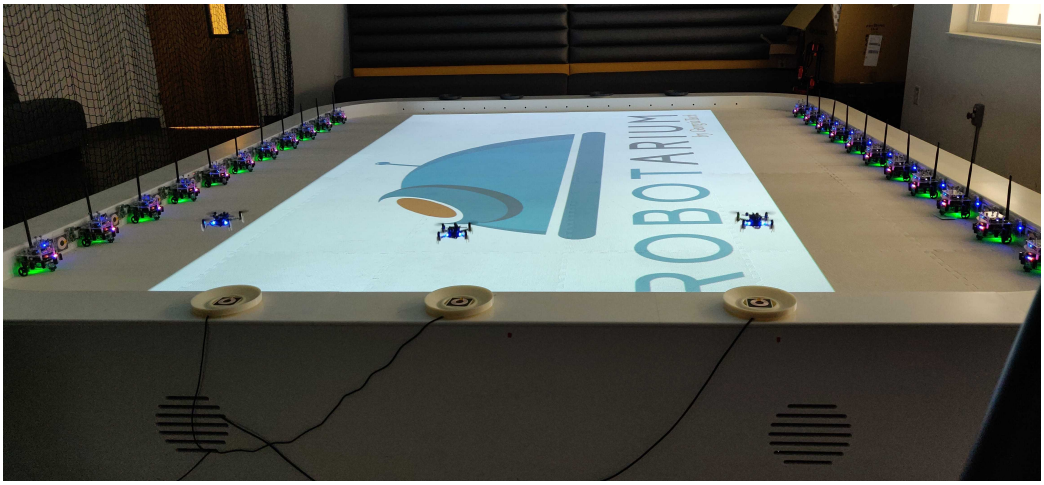


Figure 7.3: After each experiment, the autonomous charging routine is called for all active quadrotors to return to their designated charging pads.

7.3 Safety

In this section, we provide details on the safety considerations implemented on the Robotarium to ensure user experiments are safe, i.e. do not damage the equipment and quadrotor interactions remain collision-free. We provide a number of safeguards to guarantee safety such as implementing control barrier certificates and providing firmware updates.

7.3.1 Exponential Barrier Certificates

Control barrier certificates guarantee provably collision-free interactions for all aerial vehicles in the Robotarium. Also, they are designed to affect user inputs only if collisions will occur from the unmodified input. Control barrier certificates are enforced through control barrier functions (CBFs). CBFs are Lyaponuv-like functions which can be used to guarantee the forward invariance of a desired set, i.e. if the system starts in the safe set, it stays in the safe set for all time. This can be used to ensure collision-free flight maneuvers, introduced in [78] through the use of differential flatness for safe trajectory generation.

Consider the dynamics of the integrator system in Section 7.1.3. Due to the high relative degree of the output $y = q$, we leverage exponential control barrier functions (ECBF) [134].

[1] https://www.bitcraze.io/products/crazyflie-2-1/

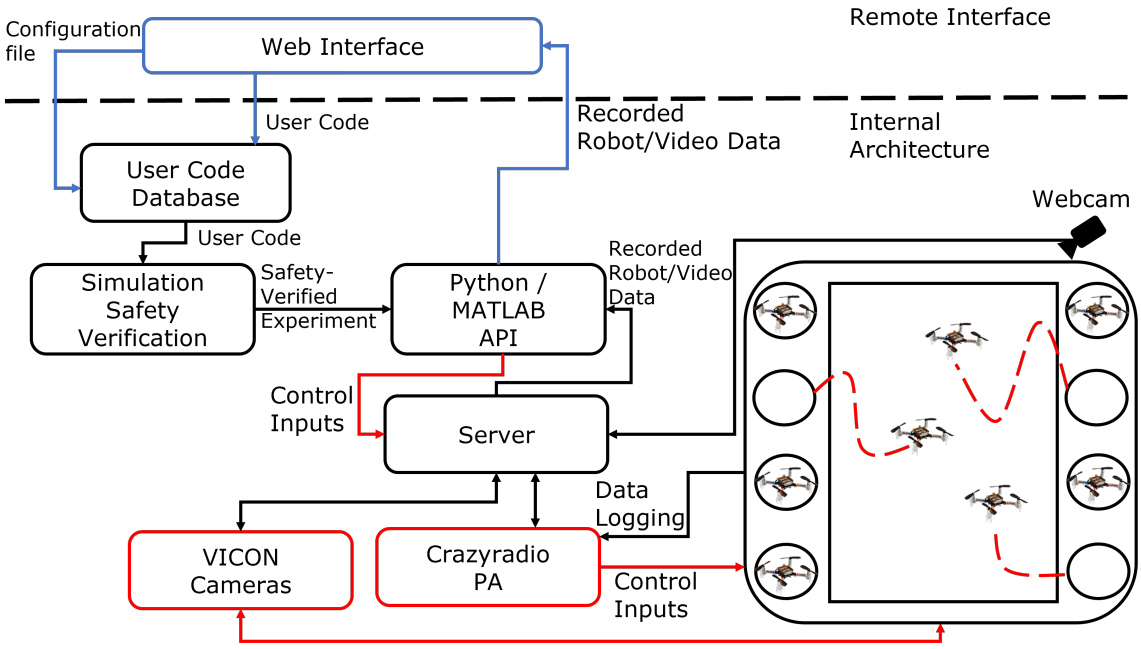


Figure 7.4: Architecture Overview. Users submit code and configuration files through a web interface. The configuration file contains the desired number of quadrotors, a Boolean check for internal sensor data logging and the desired run-time; if no file is present, the default values are used. Internally, experiments are checked for safety violations and control inputs are generated via a Python or MATLAB API. These control inputs are then sent via radio to the Crazyflies. Components directly accessible to the user are blue, Components that interface with the quadrotors are red and all other components that interact with the host PC are marked in black.

Next, consider a team of N quadrotors with dynamics modelled as fourth-order integrators with virtual inputs $v_i \in \mathbb{R}^3$,

$$\ddddot{r}_i = v_i \tag{7.5}$$

where $r_i = [x_i, y_i, z_i]^T$ is the position of the center of mass (CoM) of quadrotor i and the full state is $q = [r^T, \dot{r}^T, \ddot{r}^T, \dddot{r}^T]^T \in \mathbb{R}^{12}$. Let $r = [r_1^T, r_2^T, \ldots, r_N^T]^T \in \mathbb{R}^{3N}$ and $v = [v_1^T, v_2^T, \ldots, v_N^T]^T \in \mathbb{R}^{3N}$ denote the ensemble variables for position and virtual control of the team of quadrotors.

The following CBF ensures pairwise safety between quadrotors i and j

$$h_{ij}^1(q_i, q_j) = (x_i - x_j)^4 + (y_i - y_j)^4 + \left(\frac{z_i - z_j}{c}\right)^4 - D_s^4 \tag{7.6}$$

where D_s is the safety distance and c is the scaling factor along the z axis caused by air flow disturbance. Indeed, safe trajectories are guaranteed by encapsulating each quadrotor in a rectangular super-ellipsoid to ensure quadrotors maintain a safe distance in the $x-y$ plane as well as on the $z-$axis. Collision-free trajectories are ensured when $h^1_{ij} \geq 0$.

From [78], the virtual input v must satisfy

$$-\left[0,\ldots,\underbrace{1}_{i^{th}},\ldots,\underbrace{-1}_{j^{th}},\ldots,0\right]\otimes\boldsymbol{\Lambda}\cdot v \tag{7.7}$$

$$\leq \boldsymbol{\gamma}\boldsymbol{K}\cdot\boldsymbol{\eta}+L^4_f h^1_{ij}(q)\,\forall i\neq j \tag{7.8}$$

where $\boldsymbol{\Lambda}$ is $[4(x_i-x_j)^3, 4(y_i-y_j)^3, 4\frac{(z_i-z_j)^3}{c^4}]$, $\boldsymbol{\eta}=[h^1_{ij},\dot{h}^1_{ij},\ddot{h}^1_{ij},\dddot{h}^1_{ij}]^T$, $K\in\mathbb{R}^{1\times 4}$ is a vector obtained by pole placement for a closed-loop matrix $(F-GK)$ and $L^4_f h^1_{ij} = 24\dot{\delta}^4+144\delta\circ\dot{\delta}^2\circ\ddot{\delta}+36r^2\circ\ddot{\delta}^2+48r^2\circ\dot{\delta}\circ\dddot{\delta})\cdot\mathbf{1}_3$ for $\delta=[x_i-x_j, y_i-y_j, \frac{z_i-z_j}{c}]$ and $\circ$ is the element-wise vector product.

The given inequality can be treated as a linear constraint on v where $A_{ij}\cdot v$ is represented by Equation 7.7 and b_{ij} is Equation 7.8. These constraints form the convex set K_{safe} for all inputs v that provide pairwise safety barrier constraints where

$$K_{safe}=\{v\in\mathbb{R}^{3N}|A_{ij}\cdot v\leq b_{ij},\forall i\neq j, j\in N\} \tag{7.9}$$

As long as the virtual control v is in K_{safe} for its corresponding initial conditions, the team of quadrotors is guaranteed safe. Next, we will show a similar formulation for the boundary of the arena.

7.3.2 Arena Barrier Certificates

In addition to providing safety guarantees for robot collision avoidance, we also provide testbed collision avoidance. We consider the CBF

$$h_i^2 = -(x_i - B_x)^4 - (y_i - B_y)^4 - (z_i - B_z)^4 \tag{7.10}$$

$$+ (\boldsymbol{B}_{max} - \boldsymbol{B}_{cent}) - D_B^4 \tag{7.11}$$

for quadrotor i where the desired closest boundary distance is D_B, $\boldsymbol{B}_{cent} = [B_x, B_y, B_z]$ are the coordinates for the center of the boundary volume and $\boldsymbol{B}_{max}$ are the maximum $x-y-z$ coordinates of the boundary. Through a similar formulation as the previous section, we use this CBF as a linear constraint for each quadrotor where

$$A_i = -diag(4(x_i - B_x)^3, 4(y_i - B_y)^3, 4(z_i - B_z)^4) \tag{7.12}$$

$$b_i = \gamma \boldsymbol{K} + L_f^4 h_i^2(q) \tag{7.13}$$

where the barrier inequality constraint for quadrotor i has the form

$$A_i v \leq b_i.$$

These previously defined constraints are combined to ensure a guaranteed safe and minimally invasive controller via the following quadratic program (QP)

$$v^* = \underset{v \in \mathbb{R}^{3N}}{\text{argmin}} \quad J(u) = \sum_{i=1}^{N} ||v_i - v_{nom}||^2 \tag{7.14}$$

$$\text{s.t. } A_{ij} v \leq b_{ij} \quad \forall i \neq j, j \in N \tag{7.15}$$

$$A_i v_i \leq b_i \quad \forall i \in N \tag{7.16}$$

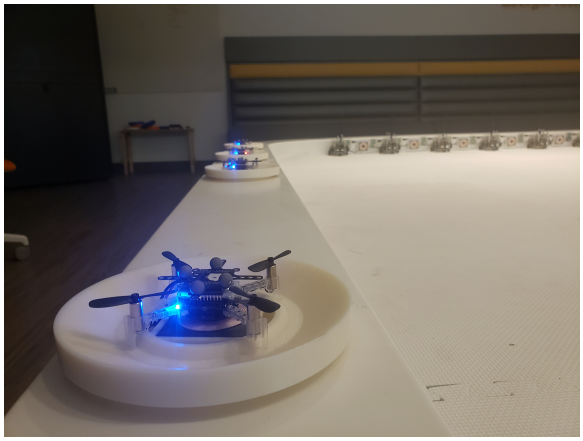


Figure 7.5: Charging docks are 3D printed in a funnel-like shape to aid landing procedures. Each charging dock contains a Qi charger transmitter that inductively charges Crazyflies.

where v_{nom} is the control input generated from the spline developed by the user's waypoint and v^* is the actual control command. The solution provided by this QP guarantees that trajectories generated for quadrotors remain collision free both for the arena and between quadrotors.

7.3.3 Firmware Updates

Safe trajectories greatly enhance the long term use of the testbed by ensuring multiple user experiments can be run without damaging the quadrotors or arena. However, some user code can be unintentionally malicious through algorithm design. We therefore update the open source firmware of the Crazyflie by implementing a switching controller design. We use the Mellinger controller [5] for all normal operations of flight; however, in the event that no message packet is recieved from the Crazyflie Realtime Protocol (CRTP), we induce a hover controller. This hover controller ensures that if a user's code blocks the ROS topic responsible for sending new input commands, the quadrotor will not fall while waiting for new inputs to be received. If the next command takes longer than a predefined number of seconds, the quadrotors will land automatically and wait for the next experiment.

7.4 Autonomous Charging

A key addition to the inclusion of quadrotors on the Robotarium is the ability to land and charge quadrotors autonomously. Funnel-like charging docks are 3D printed (see Figure 7.5) that leverage the grounding effects [135] of the propellers to assist quadrotor landing maneuvers. A key goal of autonomous landing and charging is to enable repeatable take-off and landing from the charging docks and to return to the charging docks and resume charging the quadrotors from anywhere on the testbed. In this section, we will introduce two procedures that enable landing and charging for external user experiments.

Algorithm 7 Autonomous Landing

$set\_desired\_poses \leftarrow post\_takeoff\_poses$
$update\_poses()$
$start \rightarrow post\_takeoff\_poses_z$
$end \rightarrow pre\_takeoff\_poses_z$
$\eta \rightarrow desired\_trajectories(start, end)$
while $r \neq pre\_takeoff\_poses$ **do**
    $cmdFullState(\eta_{1:9}, \psi, {}_B\dot{\omega})$
$notifySetpointsStop()$
$land()$

Our first procedure is the **land()** function with the algorithm shown in Algorithm 7. The software stack first saves the post-takeoff and pre-takeoff poses of all quadrotors in a user experiment. Then we set desired poses to be the post-takeoff positions (positions of quadrotors at the start of the experiment), once reached, we define a new set of waypoints defined by intermediate heights between the post-takeoff and pre-takeoff pose heights, labeled as $post\_takeoff\_poses_z$ and $pre\_takeoff\_poses_z$. Trajectories are generated for all active quadrotors and the desired setpoint is developed by the chain of integrator system and sent as a ROS topic to *cmdFullState*. In the landing procedure, the arena barrier certificates are not enabled as the docking stations are outside of the boundary volume. Finally, after the quadrotors have landed we call the *notifySetpointsStop* service in the Crazyswarm ROS package which halts all streaming setpoints passing from the host computer to the

Crazyflies.

The second procedure we implement is the **charge_robots()** procedure. We first enable logging in the Crazyswarm launch file, allowing us to record realtime log data provided by the vendor firmware and access the charge status of the crazyflies. Crazyflie battery states include:

- Battery: 0
- Charging: 1
- Charged: 2
- Low power: 3
- Shutdown: 4

We first check if the quadrotor is not in the charging or charged state and if the battery state is not one of these values, the **charge_robots()** script begins. We implement an aggressive trajectory tracking controller to adjust position due to small changes in position error for all charging routines. The charging locations, previously recorded via a configuration file loaded at the beginning of the experiment, are set as the desired endpoints for each quadrotor trajectory. Each quadrotor is landed independently of the first to ensure external disturbances do not impact landing procedures. Once all robots have entered the charged or charging state, the autonomous charging procedure exits.

7.5 Software, Simulation and Safety Verification

The software addition to the Robotarium was designed to ease user access to aerial swarms for robotics research. We developed our software stack with that goal by introducing: model-based simulation, data logging and safety verification.

The simulator uses model-based simulation for users to test their algorithms and see how their code would operate on real world systems before implementation. The model

and model parameters are available in the simulator for users to see and adjust their code accordingly. Users must first declare the number of quadrotors desired for an experiment and then call the **build()** function which initializes the experiment and generates the simulation window. Interaction with the robots is limited to waypoint control through the **set_desired_poses()** function where users give an array of desired positions for their swarm. Finally, these pose commands are updated using the **update_poses()** function which updates the dynamical model of the quadrotors and simulates the quadrotors motion via the provided waypoints. The simulator is available as a MATLAB simulator or Python simulator on the Robotarium GitHub [2].

Often user experiments require data recorded from the experiments, users can save experiment data via the **save_data()** function. This function generates a pickle file (or .mat file) with a record of the quadrotor position, chain of integrator input, and orientation for a particular timestamp. These values are replaced with the real values in the submitted experiments.

Finally, we implement a safety verification checker via simulation when users submit experiments. This checks how often we modify user input to ensure safety and experiments where a high number of collisions are rejected based on a predefined threshold D_{safe}. This is represented by the function $D \in \mathbb{R}^+$ for N robots over time $t \in [0, T]$. We represent this safety check formally as

$$D = \sum_{i=0}^{N} D_i = \sum_{t=0}^{T} I_i(t)dt \tag{7.17}$$

where $I_i(t)$ is an indicator function that is 1 if a quadrotor has "collided" at time t and 0 otherwise. For the entire user experiment $D \leq D_{safe}$ to satisfy the virtual safety checker.

Table 7.1: Videos of Quadrotor Experiments

Experiment	Video Links
Adversarial Agent	https://youtu.be/XfVc4Yp6qiM
Circle Swap	https://youtu.be/xTRfJkU3MEE
Leader Follower	https://youtu.be/VPc6ooAvHUU

7.6 Types of Experiments

In this section we show examples of the types of experiments we have run using the aerial vehicle framework. These experiments were first simulated and then ran autonomously on our testbed. All corresponding videos for each experiment can be found in Table 7.1.

7.6.1 Adversarial Agent Interactions with Exponential Barrier Certificates

This experiment is a demonstration of the effectiveness of barrier functions implemented on quadrotors [78]. This shows four quadrotors in a circle formation with an antagonistic fifth quadrotor flying through the formation. Because barrier functions are implemented on all quadrotors, as the fifth quadrotor tries to collide with the other quadrotors, each trajectory is modified and all quadrotors remain safe and collision-free.

7.6.2 Swap Positions in Three-Dimensional Space

A classic example of barrier functions in two-dimensions for the GritsBots is the swap positions experiment [113]. In this example, each robot is commanded to swap positions with its furthest neighbor in a circle. Emergent collision avoidance behavior is generated through safe trajectories created through control barrier certificates. We implement this on the quadrotors and show this emergent behavior in three-dimensional space.

7.6.3 Leader Follower Network Control on Undirected Weighted Graphs

Many swarm robotics applications develop leader-follower like networks where a single or a subset of agents influence the behavior of a group of agents. These types of networks

[2]https://github.com/robotarium

can be seen in biomemetic robotics applications [136], military operations [137] or sensor network emulations [138]. We show through the development of a leader follower network on undirected weighted graphs, first presented in [139], the extension of leader-follower networks for aerial swarms represented by the dynamics

$$\dot{x} = -((Diag(\hat{\delta}_f)L_w) \otimes I_d)x + (\Delta_l \otimes I_d)u_l \tag{7.18}$$

where $Diag(\hat{\delta}_f)$ is the diagonal indicator matrix representing which agents are followers, L_w is the weighted graph Laplacian of graph G, I_d is a $d \times d$ identity matrix, Δ_d is the matrix representing the leaders in G, u_l is the input vector to the leaders and $x \in \mathbb{R}^d$. This system shows that easy prototyping of network control type problems can be created in simulation and verified in experiment on our platform.

7.7 Conclusion

In conclusion, we have developed a remotely accessible aerial swarm robotics testbed as an addition and improvement upon the original Robotarium. We highlight the software and hardware changes made to enable safe deployment of external user experiments. We provide formal guarantees of safety through the introduction of control barrier certificates for quadrotors and safety verification by measuring collision information. To show the efficacy of our testbed we show a number of examples created to highlight the versatility of swarm robotics applications that quadrotors are capable of executing while remaining collision free and autonomous.

CHAPTER 8
CONCLUSIONS AND FUTURE WORKS

As humans begin working more frequently in environments with multi-agent systems, they are presented with challenges on how to control these systems in an intuitive manner. It is often the case that to control robotic systems, experts must design complex controllers for single agents or provide basic trajectory commands for the multi-agent case. We seek to democratize the use of robotic systems where novice users can provide high level specifications with little overhead necessary to achieve desired goals.

In Chapter 3 we begin by exploring *specification-based maneuvering* where we leverage the dynamics of quadrotors to design trajectories that satisfy goal specifications as well as the dynamical constraints of the system. In addition to this, we show how to develop specifications both as explicit sequences of hoops as well as LTL specifications. By including LTL specifications we enhance the expressive capabilities of our planning framework for reconfigurable waypoints.

In Chapter 4 we define *task orchestration* as a combination of task decomposition, allocation and planning for a quadrotor or team of quadrotors given a high-level specification. By doing this we consider tasks that are complex and consist of environment constraints, system constraints, or both, that must be satisfied. Trajectories are optimized according to agent specific cost functions for a homogeneous team of agents (in our case quadrotors). We then verify this task orchestration framework with a fire-fighting quadrotors case study. This problem is complex enough in that it captures many aspects of the types of problems this orchestration framework seeks to solve. Indeed, by considering a set of N quadrotors, capable of carrying water, surveying various locations and identifying resources within a predefined area, we dynamically allocate these quadrotors to different regions, extinguish fires and monitor their internal states in an efficient manner.

This task decomposition is extended in Chapter 5 where we define an online method of task allocation for multi-agent systems. We develop online cross entropy optimization for online task allocation and consider use cases where end users are unfamiliar with optimal costs necessary in the prior problem formulation of Chapter 4. In addition to this, we extended online cross entropy to consider multivariate distribution sampling and its applications to robotic systems.

In the prior parts of this thesis, we assumed end users are familiar with LTL specifications. And while this formal language is expressive and close to human syntax, often experts are needed to understand ideal construction of specifications for preferred behavior of robotic systems. With this in mind, we approach learning temporal logic specifications through support vector machines in Chapter 6. Features are learned from traces and then mapped to temporal logic templates where specifications are generated as a composition of temporal logic templates. This enables novice users to "define" desired system specifications by only providing traces to a learning algorithm. We validate this work with a comparison to brute force LTL mining techniques as well as a case study involving home surveillance drones.

Throughout this thesis, specification-based planning has been used to provide end users control to aerial swarms. In Chapter 7, we provide external users access to waypoint control for real hardware systems via the Robotarium. In addition to this, the extension of the Robotarium aims to bridge the hardware-software gap prevalent in multi-robot research and aerial swarm robotic research by providing an open access research platform to users with limited knowledge of aerial vehicles or testbeds. We present the hardware and software changes necessary to provide users safe access to micro-UAVs and provide examples of the types of experiments users can submit to the Robotarium.

In conclusion, this thesis provides intuitive methods of quadrotor control for the single and multi-agent cases. We leverage system and environment constraints to inform planning and task decomposition with minimal input from the end users. It is our belief that

by using specification-based planning, novice users can define and satisfy complex goals for quadrotor teams. It is evident from the use of formal methods, such as temporal logic, for specification design that we can guarantee system performance and specification design through the tools we have presented in this thesis. Additionally, we show that we further reduce obstacles for robot control for novice users using specification design by learning temporal logic specifications. It is here we envision future research can be approached to leverage additional information from a system, or desired behavior, to generate specifications defining its generalized behavior formally through temporal logics.

VITA

Christopher Banks was born in Portsmouth, Virginia, on September 28, 1995. He graduated from Churchland High School with Honors in June 2013. He received a B.S. in Physics from Norfolk State University in 2017 and a M.S. in Computer Science from the Georgia Institute of Technology in 2021. He has been awarded the National Science Foundation Graduate Research Fellowship in 2017.

REFERENCES

[1] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, "Human interaction with robot swarms: A survey," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 9–26, 2015.

[2] A. Kolling, K. Sycara, S. Nunnally, and M. Lewis, "Human-swarm interaction: An experimental study of two types of interaction with foraging swarms," *Journal of Human-Robot Interaction*, vol. 2, no. 2, pp. 103–129, 2013.

[3] D. Szafir, B. Mutlu, and T. Fong, "Designing planning and control interfaces to support user collaboration with flying robots," *The International Journal of Robotics Research*, vol. 36, no. 5-7, pp. 514–542, 2017.

[4] Y. Wang, A. Ramirez-Jaime, F. Xu, and V. Puig, "Nonlinear model predictive control with constraint satisfactions for a quadcopter," *Journal of Physics: Conference Series*, vol. 783, p. 012 025, Jan. 2017.

[5] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 2520–2525.

[6] D. Aksaray, C.-I. Vasile, and C. Belta, "Dynamic routing of energy-aware vehicles with temporal logic constraints," in *Proc. IEEE ICRA 2016*, Stockholm, Sweden, May 2016, pp. 3141–3146.

[7] J. O. Huckaby and H. I. Christensen, "A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics," in *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[8] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[9] A. Camacho and S. A. McIlraith, "Learning interpretable models expressed in linear temporal logic," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 621–630.

[10] A. Shah, P. Kamath, J. A. Shah, and S. Li, "Bayesian inference of temporal task specifications from demonstrations," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[11] L. Consolini, F. Morbidi, D. Prattichizzo, and M. Tosques, "Leader–follower formation control of nonholonomic mobile robots with input constraints," *Automatica*, vol. 44, no. 5, pp. 1343–1349, 2008.

[12] A. Y. Yazicioğlu and M. Egerstedt, "Leader selection and network assembly for controllability of leader-follower networks," in *2013 American Control Conference*, IEEE, 2013, pp. 3802–3807.

[13] T. Setter, H. Kawashima, and M. Egerstedt, "Team-level properties for haptic human-swarm interactions," in *2015 American Control Conference (ACC)*, 2015, pp. 453–458.

[14] J. Nagi, A. Giusti, L. M. Gambardella, and G. A. D. Caro, "Human-swarm interaction using spatial gestures," Institute of Electrical and Electronics Engineers Inc., Oct. 2014, pp. 3834–3841, ISBN: 9781479969340.

[15] J. Alonso-Mora, S. Haegeli Lohaus, P. Leemann, R. Siegwart, and P. Beardsley, "Gesture based human - multi-robot swarm interaction and its application to an interactive display," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 5948–5953.

[16] Z. Kira and M. A. Potter, "Exerting human control over decentralized robot swarms," 2009, pp. 566–571, ISBN: 9781424427130.

[17] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," 2005.

[18] M. Wu, G. Yan, Z. Lin, and Y. Lan, "Synthesis of output feedback control for motion planning based on ltl specifications," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2009, pp. 5071–5075.

[19] C. Zielinski and T. Winiarski, "General specification of multi-robot control system structures," *BULLETIN OF THE POLISH ACADEMY OF SCIENCES TECHNICAL SCIENCES*, vol. 58, Mar. 2010.

[20] L. Wang, E. A. Theodorou, and M. Egerstedt, "Safe learning of quadrotor dynamics using barrier certificates," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 2460–2465.

[21] S. L. Herbert, S. Bansal, S. Ghosh, and C. J. Tomlin, "Reachability-based safety guarantees using efficient initializations," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 4810–4816.

[22] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[23] C. Baier and J.-P. Kateon, *Principles of Model Checking*. Cambridge, Massachusetts: The MIT Press, 2008.

[24] A. Pnueli, "Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends," *Current trends in Concurrency*, pp. 510–584, 1986.

[25] E. M. Clarke, O. Grumberg, and D. Peled, *Model checking*. MIT press, 1999.

[26] E. Plaku, "Planning in discrete and continuous spaces: From LTL tasks to robot motions," in *Conference Towards Autonomous Robotic Systems*, Springer, 2012, pp. 331–342.

[27] Z. Xu, *LTL motion planning with collision avoidance for a team of quadrotors*, 2016.

[28] M. Kloetzer, X. C. Ding, and C. Belta, "Multi-robot deployment from ltl specifications with reduced communication," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, IEEE, 2011, pp. 4867–4872.

[29] B. Lacerda, D. Parker, and N. Hawes, "Optimal and dynamic planning for markov decision processes with co-safe ltl specifications," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1511–1516.

[30] D. Aksaray, K. Leahy, and C. Belta, "Distributed multi-agent persistent surveillance under temporal logic constraints," *IFAC-PapersOnLine*, vol. 48, no. 22, pp. 174–179, 2015.

[31] L. Luo, N. Chakraborty, and K. Sycara, "Distributed algorithm design for multi-robot task assignment with deadlines for tasks," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 3007–3013.

[32] J. Chen, S. Moarref, and H. Kress-Gazit, "Verifiable control of robotic swarm from high-level specifications," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, International Foundation for Autonomous Agents and Multiagent Systems, 2018, pp. 568–576.

[33] J. Tůmová and D. V. Dimarogonas, "Multi-agent planning under local LTL specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239–248, 2016.

[34] P. Schillinger, M. Burger, and D. V. Dimarogonas, "Decomposition of finite LTL specifications for efficient multi-agent planning," in *Distributed Autonomous Robotic Systems*, Springer, 2018, pp. 253–267.

[35] C. Lemieux, D. Park, and I. Beschastnikh, "General ltl specification mining."

[36] W. M. van der Aalst, H. De Beer, and B. F. van Dongen, "Process mining and verification of properties: An approach based on temporal logic," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, Springer, 2005, pp. 130–147.

[37] S. de Amo and D. A. Furtado, "First-order temporal pattern mining with regular expression constraints," *Data Knowledge Engineering*, vol. 62, no. 3, pp. 401–420, 2007, Including special issue: 20th Brazilian Symposium on Databases (SBBD 2005).

[38] C. Yoo and C. Belta, "Rich time series classification using temporal logic," in *Robotics: Science and Systems*, 2017.

[39] J. Kim, C. Muise, A. Shah, S. Agarwal, and J. Shah, "Bayesian inference of linear temporal logic specifications for contrastive explanations," 2019.

[40] A. Khamis, A. Hussein, and A. Elmogy, "Multi-robot task allocation: A review of the state-of-the-art," in *Cooperative Robots and Sensor Networks 2015*, Springer, 2015, pp. 31–51.

[41] D. 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.*, 2005, pp. 151–157.

[42] A. M. Khamis, A. M. Elmogy, and F. O. Karray, "Complex task allocation in mobile surveillance systems," *Journal of Intelligent & Robotic Systems*, vol. 64, no. 1, pp. 33–55, 2011.

[43] P. Skobelev, D. Budaev, N. Gusev, and G. Voschuk, "Designing multi-agent swarm of uav for precise agriculture," in *International Conference on Practical Applications of Agents and Multi-Agent Systems*, Springer, 2018, pp. 47–59.

[44] R. D'Andrea, "Guest editorial: A revolution in the warehouse: A retrospective on kiva systems and the grand challenges ahead," *IEEE Transactions on Automation Science and Engineering*, vol. 9, no. 4, pp. 638–639, 2012.

[45] M. Vaidis and M. J.-D. Otis, "Swarm robotic interactions in an open and cluttered environment: A survey," *Designs*, vol. 5, no. 2, p. 37, 2021.

[46] J.-P. D. L. Croix and M. Egerstedt, "Networks and heterogeneous media analyzing human-swarm interactions using control lyapunov functions and optimal control," vol. 10, 3 2015.

[47] T. Gustavi, D. V. Dimarogonas, M. Egerstedt, and X. Hu, "Sufficient conditions for connectivity maintenance and rendezvous in leader–follower networks," *Automatica*, vol. 46, no. 1, pp. 133–139, 2010.

[48] A. Franchi, C. Secchi, H. I. Son, H. H. Bulthoff, and P. R. Giordano, "Bilateral teleoperation of groups of mobile robots with time-varying topology," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1019–1033, 2012.

[49] H. Saeidi, D. G. Mikulski, and Y. Wang, "Trust-based leader selection for bilateral haptic teleoperation of multi-robot systems," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 6575–6581.

[50] J. Nagi, H. Ngo, L. M. Gambardella, and G. A. Di Caro, "Wisdom of the swarm for cooperative decision-making in human-swarm interaction," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 1802–1808.

[51] S. Zhao, Z. Li, R. Cui, Y. Kang, F. Sun, and R. Song, "Brain–machine interfacing-based teleoperation of multiple coordinated mobile robots," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 6, pp. 5161–5170, 2016.

[52] D. R. Frutiger, B. E. Kratochvil, K. Vollmers, and B. J. Nelson, "Magmites-wireless resonant magnetic microrobots," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, 2008, pp. 1770–1771.

[53] M. Egerstedt, J.-P. D. L. Croix, H. Kawashima, and P. Kingston, "Interacting with networks of mobile agents," 2014.

[54] P. Song and V. Kumar, "A potential field based approach to multi-robot manipulation," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, IEEE, vol. 2, 2002, pp. 1217–1222.

[55] J. L. Baxter, E. Burke, J. M. Garibaldi, and M. Norman, "Multi-robot search and rescue: A potential field based approach," in *Autonomous robots and agents*, Springer, 2007, pp. 9–16.

[56] Z. Mason, "Programming with stigmergy: Using swarms for construction," in *ICAL 2003: Proceedings of the eighth international conference on Artificial life*, 2003, pp. 371–374.

[57] J. Cortes, S. Martinez, T. Karatas, and F. Bullo, "Coverage control for mobile sensing networks," *IEEE Transactions on robotics and Automation*, vol. 20, no. 2, pp. 243–255, 2004.

[58] M. Santos, Y. Diaz-Mercado, and M. Egerstedt, "Coverage control for multirobot teams with heterogeneous sensing capabilities," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 919–925, 2018.

[59] H. Hexmoor, B. McLaughlan, and M. Baker, "Swarm control in unmanned aerial vehicles.," in *IC-AI*, 2005, pp. 911–917.

[60] A. E. Gerevini, P. Haslum, D. Long, A. Saetti, and Y. Dimopoulos, "Deterministic planning in the fifth international planning competition: Pddl3 and experimental evaluation of the planners," *Artificial Intelligence*, vol. 173, no. 5-6, pp. 619–668, 2009.

[61] P. Nilsson *et al.*, "Toward specification-guided active mars exploration for cooperative robot teams," 2018.

[62] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[63] A. Bhatia, M. R. Maly, L. E. Kavraki, and M. Y. Vardi, "Motion planning with complex goals," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 55–64, 2011.

[64] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 1478–1483.

[65] P. Fiorini and Z. Shiller, "Time optimal trajectory planning in dynamic environments," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 2, 1996, 1553–1558 vol.2.

[66] P. Reynoso-Mora, W. Chen, and M. Tomizuka, "A convex relaxation for the time-optimal trajectory planning of robotic manipulators along predetermined geometric paths," *Optimal Control Applications and Methods*, vol. 37, no. 6, pp. 1263–1281, 2016.

[67] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3, pp. 293–321, 1992.

[68] B. Bonet and H. Geffner, "Planning as heuristic search," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 5–33, 2001.

[69] S. A. Bortoff, "Path planning for UAVs," in *American Control Conference, 2000. Proceedings of the 2000*, IEEE, vol. 1, 2000, pp. 364–368.

[70] G. Hoffmann, S. Waslander, and C. Tomlin, "Quadrotor helicopter trajectory tracking control," in *AIAA guidance, navigation and control conference and exhibit*, 2008, p. 7410.

[71] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, "Quadrotor helicopter flight dynamics and control: Theory and experiment," in *AIAA guidance, navigation and control conference and exhibit*, 2007, p. 6461.

[72] T. Luukkonen, "Modelling and control of quadcopter," *Independent research project in applied mathematics, Espoo*, vol. 22, p. 22, 2011.

[73] D. Zhou and M. Schwager, "Vector field following for quadrotors using differential flatness," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, IEEE, 2014, pp. 6567–6572.

[74] H. Sira-Ramirez and S. K. Agrawal, *Differentially flat systems*. Crc Press, 2018.

[75] R. M. Murray, M. Rathinam, and W. Sluis, "Differential flatness of mechanical control systems: A catalog of prototype systems," in *ASME international mechanical engineering congress and exposition*, Citeseer, 1995.

[76] J. Ferrin, R. Leishman, R. Beard, and T. McLain, "Differential flatness based control of a rotorcraft for aggressive maneuvers," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 2688–2693.

[77] N. T. Nguyen, I. Prodan, F. Stoican, and L. Lefevre, "Reliable nonlinear control for quadcopter trajectory tracking through differential flatness," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 6971–6976, 2017.

[78] L. Wang, A. D. Ames, and M. Egerstedt, "Safe certificate-based maneuvers for teams of quadrotors using differential flatness," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 3293–3298.

[79] X. Yu, X. Zhou, K. Guo, J. Jia, L. Guo, and Y. Zhang, "Safety flight control for a quadrotor uav using differential flatness and dual-loop observers," *IEEE Transactions on Industrial Electronics*, 2021.

[80] J. Zeng, P. Kotaru, M. W. Mueller, and K. Sreenath, "Differential flatness based path planning with direct collocation on hybrid modes for a quadrotor with a cable-suspended payload," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3074–3081, 2020.

[81] A. Pnueli, "The temporal logic of programs/proceedings of the 18th ieee symposium on foundation of computer science," 1977.

[82] G. J. Holzmann, *The SPIN model checker: Primer and reference manual*. Addison-Wesley Reading, 2004, vol. 1003.

[83] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Improving multi-robot behavior using learning-based receding horizon task allocation," in *Robotics: Science and Systems (RSS)*, 2018.

[84] C. Menghi, S. Garcia, P. Pelliccione, and J. Tůmová, "Multi-robot LTL planning under uncertainty," in *International Symposium on Formal Methods*, Springer, 2018, pp. 399–417.

[85] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe ltl specifications," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 1525–1532.

[86] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[87] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[88] S. G. Loizou and K. J. Kyriakopoulos, "Automatic synthesis of multi-agent motion tasks based on ltl specifications," in *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, IEEE, vol. 1, 2004, pp. 153–158.

[89] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, 2009.

[90] C. I. Vasile and C. Belta, "Sampling-based temporal logic path planning," *CoRR*, vol. abs/1307.7263, 2013. arXiv: 1307.7263.

[91] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, "Discrete abstractions of hybrid systems," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 971–984, 2000.

[92] S. G. Loizou and K. J. Kyriakopoulos, "Automated planning of motion tasks for multi-robot systems," in *Proceedings of the 44th IEEE Conference on Decision and Control*, IEEE, 2005, pp. 78–83.

[93] M. Kobilarov, "Cross-entropy motion planning," *The International Journal of Robotics Research*, vol. 31, no. 7, pp. 855–871, 2012.

[94] P.-T. de Boer *et al.*, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 19–67, 2005.

[95] M. Laguna, A. Duarte, and R. Martı, "Hybridizing the cross-entropy method: An application to the max-cut problem," *Computers & Operations Research*, vol. 36, no. 2, pp. 487–498, 2009.

[96] K. Chepuri and T. Homem-De-Mello, "Solving the vehicle routing problem with stochastic demands using the cross-entropy method," *Annals of Operations Research*, vol. 134, no. 1, pp. 153–181, 2005.

[97] S. C. Livingston, E. M. Wolff, and R. M. Murray, "Cross-entropy temporal logic motion planning," in *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, ACM, 2015, pp. 269–278.

[98] X. Zhang, K. Wang, and W. Dai, "Multi-uavs task assignment based on fully adaptive cross-entropy algorithm," in *2021 11th International Conference on Information Science and Technology (ICIST)*, 2021, pp. 286–291.

[99] W. Li, L. Dworkin, and S. A. Seshia, "Mining assumptions for synthesis," in *Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMPCODE2011)*, IEEE, 2011, pp. 43–50.

[100] C. Lemieux, "Mining temporal properties of data invariants," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, 2015, pp. 751–753.

[101] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett, "Patterns in property specifications for finite-state verification," *Proceedings - International Conference on Software Engineering*, pp. 411–420, 1999.

[102] L. Garcıa-Pérez, M. Garcıa-Alegre, A. Ribeiro, and D. Guinea, "An agent of behaviour architecture for unmanned control of a farming vehicle," *computers and electronics in agriculture*, vol. 60, no. 1, pp. 39–48, 2008.

[103] A. Duret-Lutz and D. Poitrenaud, "Spot: An extensible model checking library using transition-based generalized bu/spl uml/chi automata," in *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings.*, IEEE, 2004, pp. 76–83.

[104] M. Duarte *et al.*, "Application of swarm robotics systems to marine environmental monitoring," in *OCEANS 2016-Shanghai*, IEEE, 2016, pp. 1–8.

[105] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[106] C. Banks, S. Wilson, S. Coogan, and M. Egerstedt, "Multi-agent task allocation using cross-entropy temporal logic optimization," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 7712–7718.

[107] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006, ISBN: 0387310738.

[108] C. Banks, K. Slovak, S. Coogan, and M. Egerstedt, "Specification-based maneuvering of quadcopters through hoops," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2019, pp. 5191–5197.

[109] I. Szita and A. Lörincz, "Online variants of the cross-entropy method," *CoRR*, vol. abs/0801.1988, 2008. arXiv: 0801.1988.

[110] L. Devroye, "Nonuniform random variate generation," *Handbooks in operations research and management science*, vol. 13, pp. 83–121, 2006.

[111] H. David and H. Nagaraja, "Order statistics wiley," *New York*, 1981.

[112] J. Dezert and C. Musso, "An efficient method for generating points uniformly distributed in hyperellipsoids," in *Proceedings of the Workshop on Estimation, Tracking and Fusion: A Tribute to Yaakov Bar-Shalom*, 2001.

[113] D. Pickem *et al.*, "The Robotarium: A remotely accessible swarm robotics research testbed," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 1699–1706.

[114] S. L. Smith, J. Tůmová, C. Belta, and D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," *The International Journal of Robotics Research*, vol. 30, no. 14, pp. 1695–1708, 2011.

[115] M. Y. Vardi, "An automata-theoretic approach to linear temporal logic," *Logics for concurrency*, pp. 238–266, 1996.

[116] E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak, "Omega-regular objectives in model-free reinforcement learning," 2018.

[117] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*, 4. Springer, 2006, vol. 4.

[118] G. De Giacomo and M. 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*, Association for Computing Machinery, 2013, pp. 854–860.

[119] G. De Giacomo, R. De Masellis, and M. Montali, "Reasoning on LTL on finite traces: Insensitivity to infiniteness," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[120] M. Sipser, *Introduction to the theory of computation*, p. 458, ISBN: 9781133187790.

[121] D. Neider and I. Gavran, "Learning linear temporal properties," in *2018 Formal Methods in Computer Aided Design (FMCAD)*, IEEE, 2018, pp. 1–10.

[122] D. P. Williamson and D. B. Shmoys, *The design of approximation algorithms*. Cambridge university press, 2011.

[123] A. Gupta, R. Krishnaswamy, A. Kumar, and D. Panigrahi, "Online and dynamic algorithms for set cover," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, 2017, pp. 537–550.

[124] A. Abboud, R. Addanki, F. Grandoni, D. Panigrahi, and B. Saha, "Dynamic set cover: Improved algorithms and lower bounds," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 114–125.

[125] V. Krátk, P. Petráček, T. Báča, and M. Saska, "An autonomous unmanned aerial vehicle system for fast exploration of large complex indoor environments," *Journal of field robotics*, vol. 38, no. 8, pp. 1036–1058, 2021.

[126] G. Cai, J. Dias, and L. Seneviratne, "A survey of small-scale unmanned aerial vehicles: Recent advances and future development trends," *Unmanned Systems*, vol. 2, no. 02, pp. 175–199, 2014.

[127] S. Wilson *et al.*, "The robotarium: Automation of a remotely accessible, multi-robot testbed," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2922–2929, 2021.

[128] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, and R. D'Andrea, "A platform for aerial robotics research and demonstration: The flying machine arena," *Mechatronics*, vol. 24, no. 1, pp. 41–54, 2014.

[129] S.-J. Chung, A. A. Paranjape, P. Dames, S. Shen, and V. Kumar, "A survey on aerial swarm robotics," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 837–855, 2018.

[130] J. P. How, B. Behihke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Systems Magazine*, vol. 28, no. 2, pp. 51–64, 2008.

[131] M. Schmittle *et al.*, "Openuav: A uav testbed for the cps and robotics community," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, IEEE, 2018, pp. 130–139.

[132] J. A. Preiss, W. Honig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 3299–3304.

[133] T. Lee, M. Leok, and N. H. McClamroch, "Control of complex maneuvers for a quadrotor uav using geometric methods on se (3)," *arXiv preprint arXiv:1003.2005*, 2010.

[134] Q. Nguyen and K. Sreenath, "Exponential control barrier functions for enforcing high relative-degree safety-critical constraints," in *2016 American Control Conference (ACC)*, IEEE, 2016, pp. 322–328.

[135] P. Sanchez-Cuevas, G. Heredia, and A. Ollero, "Characterization of the aerodynamic ground effect and its influence in multirotor control," *International Journal of Aerospace Engineering*, vol. 2017, 2017.

[136] N. Tarcai *et al.*, "Patterns, transitions and the role of leaders in the collective dynamics of a simple robotic flock," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2011, no. 04, P04010, 2011.

[137] C. Kerr, R. Jaradat, and N. U. I. Hossain, "Battlefield mapping by an unmanned aerial vehicle swarm: Applied systems engineering processes and architectural considerations from system of systems," *IEEE Access*, vol. 8, pp. 20 892–20 903, 2020.

[138] H. Wu, S. Qu, D. Xu, and C. Chen, "Precise localization and formation control of swarm robots via wireless sensor networks," *Mathematical Problems in Engineering*, vol. 2014, 2014.

[139] H. Kawashima and M. Egerstedt, "Leader selection via the manipulability of leader-follower networks," in *2012 American Control Conference (ACC)*, IEEE, 2012, pp. 6053–6058.