ABSTRACT

Title of dissertation: PLANNING, MONITORING AND LEARNING WITH SAFETY AND TEMPORAL CONSTRAINTS FOR ROBOTIC SYSTEMS

Zhenyu Lin, Doctor of Philosophy, 2019

Dissertation directed by: Professor John S. Baras
Department of Electrical and Computer Engineering

In this thesis, we address the problem of planning, monitoring and learning in robotic systems, while considering the safety and time constraints. Motion and action planning for robotic systems is important for real, physical world applications. Robots are capable of performing repetitive tasks at speeds and accuracies that far exceed those of human operators and are widely used in manufacturing, medical fields and even transportation.

Planning commonly refers to a process of converting high-level task specifications into low-level control commands that can be executed on the system of interest. Time behavior is a most important issue for the autonomous systems of interest, and it is critical for many robotic tasks. Most state of the art methods, however, are not capable of providing the framework needed for the autonomous systems to plan under finite time constraints. Safety and time constraints are two important aspects for the plan. We are interested in the safety of the plan, such as "Can the robot reach the goal without collision?". We are also interested in the time constraints for

the plan, such as "Can the robot finish this task after 3 minutes but no later than 5 minutes?". These type of tasks are important to understand in robot search and rescue or cooperative robotic production line.

In this thesis, we address these problems by two different approaches, the first one is a timed automata based approach, which focuses on a more high-level, abstracted result with less computational requirement. The other one involves converting the problem into a mixed integer linear programming (MILP) with more low-level control details but requires higher computational power. Both methods are able to automatically generate a plan that are guaranteed to be correct. The robotic systems may behave differently in runtime and not able to execute the task perfectly as planned. Given that a robotic system is naturally cyber-physical, and malfunctions can have safety consequences, monitoring the system's behavior at runtime can be key to safe operation. Therefore, it is important to consider both time and space tolerances during the planning phase, and also design runtime monitors for error detection and possible self-correction. We provide an optimization-based formulation which takes the tolerances into account, and we have designed runtime monitors to monitor the status of the systems, as well as an event-triggered model predictive controller for self-correction.

Learning is another very important aspect for the robotics field. We hope to only provide the robot with high-level task specifications, and the robot learns to accomplish the task. Thus, in the next part of this thesis, we discussed how the robot could learn to accomplish task specified by metric interval temporal logic, and how the robot could replan and self-correct if the initial plan fails to execute

correctly.

As the field of robotics is expanding from the fixed environment of a production line to complex human environments, robots are required to perform increasingly human-like manipulation tasks. Thus, for the last aspect of the thesis, we considered a manipulation task with dexterous robotic hand - Shadow Hand. We collected the multimodal haptic-vision dataset, and proposed the framework of self-assurance slippage detection and correction. We provided the simulation and also real-world implementation with a UR10 and Shadowhand robotic system.

Planning, Monitoring and Learning with Safety and Temporal Constraints for Robotic Systems

by

Zhenyu Lin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Advisory Committee:
Professor John S. Baras, Chair/Advisor
Professor Yiannis Aloimonos, Dean's Representative
Professor Cornelia Fermüller
Professor Behtash Babadi
Professor Michael Otte

Acknowledgments

This thesis becomes possible with the kind support and help of many people. I owe my gratitude to all the people who supervised, supported, encouraged and inspired me during my Ph.D. life. I will cherish the graduate study experience forever.

First and foremost, I would like to express my deepest gratitude to my advisor, Professor John S. Baras for giving me an invaluable opportunity to work on such thought-provoking and extremely interesting research problems. I am always impressed by his deep insights on the real-world problems and enthusiasm in handling challenges every time we meet. I thank him for his invaluable guidance, unstinting support, and all the inspiring advice, which motivated me to become a more independent researcher in the past four years and will continue guiding me in my future career.

I would like to thank my colleagues at the System Engineering Integration Laboratory (SEIL) and Autonomy Robotics and Cognition Lab (ARC), who have enriched my graduate life in many ways. Many thanks to Dr. Dipankar Maity, Dr. Aneesh Raghavan, Dr. Ren Mao, Dr. Wentao Luan and Dr. Yuchen Zhou who have given me a lot of advice on both research problems and industrial fields. I would like to thank my office-mate Usman A. Fiaz for all the meaningful discussions and Charles A. Meetan for working together on the robotic experiments.

I also would like to express my sincere appreciation to Dr. Behtash Babadi, Dr. Cornelia Fermuller, Dr. Yiannis Aloimonos and Dr. Michael Otte for serving on

my thesis committee and for the influential discussions and feedback. Also I would like to thank Mrs. Kim Edwards for her great administrative support.

I would like to express my deepest gratitude to my parents and other family members for the constant support, encouragement and acknowledgment.

Lastly, I would like to acknowledge the support offered by NSF grant CNS-1544787, DARPA through ARO grant W911NF1410384, AnthroTronix grant UMD-09142016, and ONR grant N00014-17-1-2622.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Automated generation of behaviors not only receives increasing attention, the ability to flexibly plan intelligent behavior can indeed significantly improve the quality of numerous systems involving autonomous agents.

In this thesis, we propose five research problems to explore planning, monitoring and learning in autonomous Systems, while considering the safety and timed temporal logic constraints.

In the first problem, we consider the automated generation of behaviors for a robotic manipulator with time constraints. Consider a task where a humanoid robot has to pick up several objects from a shelf and place them on a dining table in a certain amount of time. This task requires planning and control at several stages. The robot has to grasp the objects, manipulate them in a correct order while avoiding obstacles, and place them on a correct surface within a certain time interval. However, performing this task with one low-level controller specification is impractical. It is important to figure out the correct sequence of subtasks given the timed constraint for each task. We propose a timed automata based approach for manipulator planning, using metric interval temporal logic (MITL). We first construct the automaton model for the environment, the robot, and the agent (which

is essentially the product automaton of the environment and the robot). Then the task specification is given in MITL formula, and is transformed into timed automata. Finally we took the product of the agent and the timed automata and an optimal timed path is then synthesized using the UPPAAL verification tool.

The second problem, due to the uncertainty in the environment, the verification results obtained with respect to the system and environment models at design-time might not be transferable to the system behavior at run time. For autonomous systems operating in dynamic environments, safety of motion and collision avoidance are critical requirements. We propose a two-phase process for our safety monitoring problem. In the design phase, we obtain an execution sequence for the robot which satisfies some desired specifications and has correctness guarantee. For the runtime phase, we model the robot as a hybrid system and we build a model monitor to check whether the execution sequence at runtime matches the desired execution sequence, and a safety monitor to check the runtime safety specifications of the system.

The third problem, we present a modular Q-learning framework to deal with the robot task planning, runtime monitoring and self-correction problem. The task is specified using metric interval temporal logic (MITL) with finite time constraints. We first construct a runtime monitor automaton using three-valued LTL (LTL3), and a sub-task MITL monitor is constructed by decomposing and augmenting the monitor automaton. During the learning phase, a modular Q-learning approach is proposed such that each module could learn different sub-tasks. During runtime, the sub-task MITL monitors could monitor the execution and guide the agent for

possible self-correction if an error occurs.

For the forth problem, we present a generic optimization based method for the motion planning problem with robotic agents. We consider both the space tolerance and time tolerance for the motion planning under signal temporal logic (STL) constraints, and our goal is to maximize the combined space and time tolerances. We provide a way to generate the exact control commands, and also consider tolerances by formulating a MILP problem. An event-trigger model predictive controller (MPC) has also been designed for self-corrections during the runtime.

Finally, for real-world implementation, We provide our framework of self-monitoring and self-correction for the slippage prediction and correction problem with the UR10-Shadow Hand robotic system. When grasping an object, humans are able to prevent the object from slipping from their grasp by constantly adjusting their grip. This is possible due to our highly sensitive slip detection capabilities. However, objects slipping from the grasp of a robotic hand is difficult to detect and correct. We take advantage of both the haptic data from Biotac sensors and the synchronized video data from a camera to apply a statistical based approach to slip prediction and correction, and the ability to classify objects as either rigid or soft in order to prevent over exertion of the robotic grasp.

1.1 Main contributions

This thesis focuses on trust autonomy and the main contributions of this thesis are the followings:

- We provide solutions for automated generation of behaviors for a robotic manipulator, such that the manipulator is able to accomplish given tasks under time constraints. Two different methods are proposed, the first method is based on translating the system model and task specifications into timed-automata and the second method formulates the task planning problem into a mixed-integer-linear-programming (MILP) problem.

- We propose the design of runtime safety monitors, such that the robot is able to detect when the runtime execution deviates from the planning phase.

- We provide a learning framework for the robotic agent, such that it learns to satisfy tasks with time constraints.

- We consider both space and time tolerances during the planning, and we design a event-trigger model predictive controller to ensure the specifications are not violated.

- We collect and arrange the multimodal haptic-vision dataset, and proposed the framework of self-assurance slippage detection and correction. We provided the simulation and also real-world implementation with a UR10 and Shadowhand robotic system.

1.2 Thesis Organization

The chapters of this thesis were written such that they can be read independently. Chapter 2 considers the automated generation of behaviors for a robotic

manipulator with time constraints using a timed automata based approach. Chapter 3 discusses the design of model monitor and safety monitor to track the runtime executions. Chapter 4 discusses our modular Q-learning framework with temporal logic specifications under finite time constraints. Chapter 5 proposes an optimal setup for robot planning while considering both space and time tolerances. In Chapter 6 we propose a slippage prediction and correction framework with dexterous robotic hand. The implementation on a UR10-Shadowhand robotic system is discussed. We conclude the thesis in Chapter 7.

Chapter 2: Timed Automata Approach for Manipulator Planning

Motion and action planning for robotic manipulator is important for real, physical world applications. Consider a task where a humanoid robot has to pick up several objects from a shelf and place them on a dining table. This task requires planning and control at several stages. The robot has to grasp the objects, manipulate them in a correct order while avoiding obstacles, and place them on a correct surface. However, performing this task with one low-level controller specification is impractical. It is important to figure out the correct sequence of subtasks. In this section, we will consider the problem of automated generation of behaviors for a robotic manipulator with time constraints. We begin by a review of related works and then put forward our improvement ideas.

2.1 Related Work

Over the past decade, automated synthesis of correct-by-design controllers complying with complex behavior specifications for robot planning has attracted a great deal of research. Classical Boolean logic allows to formulate specifications ϕ over a set of propositions Π. Each proposition $\pi \in \Pi$ can either be true or false. Linear Temporal Logic (LTL) extends the Boolean logic by operators with tem-

poral semantics. Consequently, specifications can be expressed over a sequence of propositions instead of a single set. This additional expressiveness makes temporal logics useful for specifying the desired behavior of autonomous systems, effectively capturing the temporal evolution of these systems. This also provides a basis to effectively plan valid actions of a system by combining the automaton of the specification with the system model, for example in [14] [15] [8]. A hierarchical procedure to address planning under LTL specification is described as follows: the robot dynamics is abstracted into a finite, discrete transition system, a discrete plan that meets the specification is synthesized and next translated into a controller for the original system. LTL specification does not emphasize on time constraints. For real applications, a robot might be required to perform a specific task within a certain time bound, rather than at some arbitrary time in the future. Time bounded motion planning has been done in heuristic ways [17] and also by using mixed integer linear programming (MILP) framework [16]. MITL, a modification of Metric Temporal Logics (MTL), disallows the punctuation in the temporal interval, so that the left boundary and the right boundary have to be different. In general the complexity of model checking for MTL related logic is higher than that of LTL. The theoretical model checking complexity for LTL is PSPACE-complete [18]. The algorithm that has been implemented is exponential to the size of the formula. MTL by itself is undecidable. The model checking process of MITL includes transforming it into a timed automaton [5] [3]. Most aforementioned work considers the case of planning for mobile robot, and the time constraint is only considered when the position changes. In this chapter, we will consider the automated generation of behaviors for

a robotic manipulator with both time constraints for position changes (move from position A to position B) and time constraints for performing actions (grasping an object, releasing an object). We propose a timed automata based approach for manipulator planning, using metric interval temporal logic (MITL). We first construct the automaton model for the environment, the robot, and the agent (which is essentially the product automaton of the environment and the robot). Then the task specification is given in MITL formula, and is transformed into timed automata. Finally we took the product of the agent and the timed automata and an optimal timed path is then synthesized using the UPPAAL verification tool. A part of this chapter is published in [1].

2.2 Preliminaries

Definition 2.1. *(Linear Temporal Logic) The syntax of LTL formulas are defined according to the following grammar rules:*

$$\phi ::= T \,|\, \pi \,|\, \neg\phi_1 \,|\, \phi_1 \wedge \phi_2 \,|\, \bigcirc\phi_1 \,|\, \phi_1 \,\mathcal{U}\, \phi_2 \,|\, \phi_1 \,\mathcal{R}\, \phi_2 \tag{2.1}$$

With LTL formulas ϕ_1, ϕ_2, propositions $\pi \in \Pi$, and the Boolean constant T "true". The syntax includes the Boolean operators $\neg$ "not" and $\wedge$ "and", as well as the temporal operator $\bigcirc$ "next", $\mathcal{U}$ "until", and "release". The following derived operators are defined to extend the above operators:

- "or": $\phi_1 \vee \phi_2 := \neg(\neg\phi_1 \wedge \neg\phi_2)$

- "implies": $\phi_1 \rightarrow \phi_2 := \neg\phi_1 \vee \phi_2$
- "eventually": $\Diamond\phi_1 := \top\, \mathcal{U}\, \phi_1$
- "always": $\Box\phi_1 := \bot\, \mathcal{R}\, \phi_1$

LTL formulas are defined over a sequence σ of atomic propositions Π. At each discrete point in time t $\in$ N, a set of propositions $\pi \in \Pi$ given by $\sigma(t)$ denotes which propositions are true $\pi \in \sigma(t)$ or false $\pi \notin \sigma(t)$. A sequence is called finite if it is bounded by a maximum time T and then we say that the sequence has length T.

Definition 2.2. *(Metric Interval Temporal Logic)*

The syntax of MITL formulas are defined according to the following grammar rules:

$$\phi ::= \top \,|\, \pi \,|\, \neg\phi_1 \,|\, \phi_1 \wedge \phi_2 \,|\, \bigcirc\phi_1 \,|\, \phi_1\, \mathcal{U}_I\, \phi_2 \,| \tag{2.2}$$

where e I $\subseteq$ [0, ∞]. $\mathcal{U}_I$ symbolizes the timed Until operator. Sometimes we will represent $\mathcal{U}_{[0,\infty]}$ by $\mathcal{U}$. Other Boolean and temporal operators such as conjunction ($\vee$), eventually within I ($\Diamond_I$), always on I ($\Box_I$) etc. can be represented using the grammar described in the definition. For example, we can express time constrained eventually operator $\Diamond_I\phi \equiv \mathrm{T}\, \mathcal{U}_I\phi$ and so on. In this section, all the untimed temporal logic here is transformed into until operator and all the timed operator is transformed to eventually within I, to make it easier to generate a timed automaton.

- $\sigma(t) \models \pi$ iff $\pi \in \sigma(t)$
- $\sigma(t) \models \neg\phi_1$ iff $\sigma(t) \not\models \phi_1$

- $\sigma(t) \models \phi_1 \wedge \phi_2$ iff $\sigma(t) \models \phi_1$ and $\sigma(t) \models \phi_2$
- $\sigma(t) \models \circ\phi_1$ iff $\sigma(t+1) \models \phi_1$
- $\sigma(t) \models \phi_1 \, \mathcal{U}_I \, \phi_2$ iff $\exists \; s \in I$ such that $\sigma(t+s) \models \phi_2$ and $\forall s' \leq s$ it holds that $\sigma(t+s') \models \phi_1$

Definition 2.3. *(Environment Model). The environment model is given by a labeled transition system defined as*

$$L = (V_L, E_L, \Pi_L, \wedge_L)$$

consisting of

(1) a set of vertices V_L

(2) a set of edges between locations $E_L \subseteq V_L \times V_L$

(3) a set of atomic propositions Π_L

(4) a labeling function $\wedge_L$*:* $V_L \to \{0, 1\}$

(5) transition conditions δ*:* $Q \to \alpha$

Vertices V_L represent locations of interest in the environment. For example, a vertex can denote an area or specific position. The edges E_L represent navigation actions in order to change the location to a different one. An edge exists between two vertices if there is a navigation action between the respective locations such that the transition condition is satisfied.

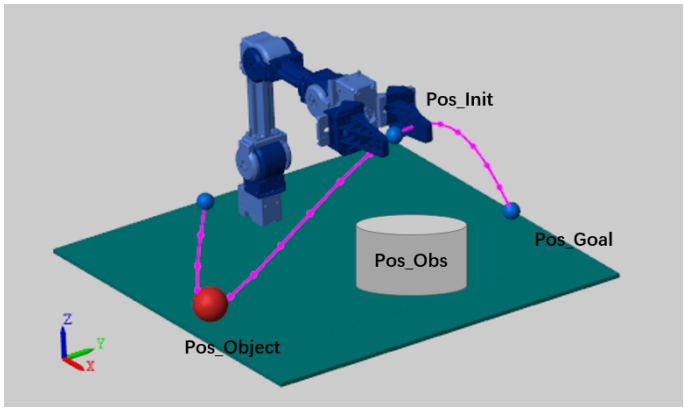


Figure 2.1: Environment Model in Simulation

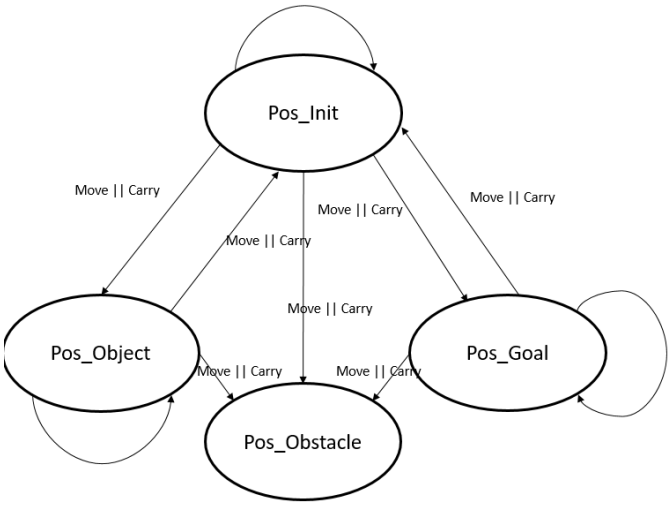


Figure 2.2: Environment Model

A simple location graph is considered for now, which includes four different locations, which are initial location, object location, goal location and an obstacle location. The transition between different nodes can happen only when the action is "move" or "carry", and it will stay at current position when the action is "hold".

Definition 2.4. *(Robot Model). The manipulator model is given by a labeled transition system defined as*

$$R = (S_R, A_R, \Pi_R, \wedge_R)$$

consisting of

(1) a set of states S_R of the robot,

(2) a set of available actions $A_R \subseteq S_R \times S_R$

(3) a set of Boolean formulas $\alpha = \alpha_i \in \alpha^{(i)}$

(4) transition conditions δ: $Q \times Q \to \alpha$ defined below.

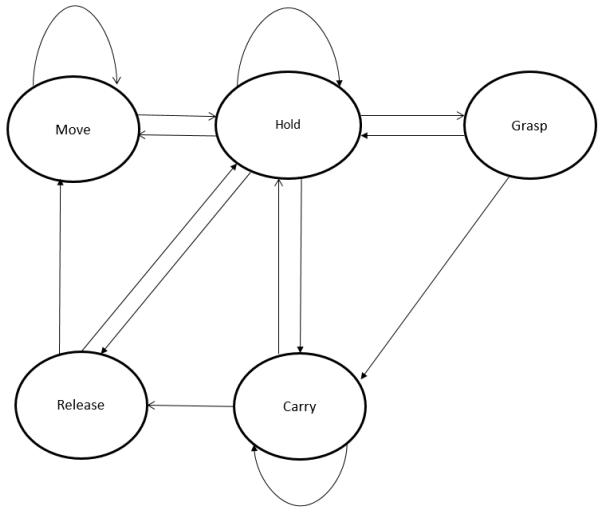


Figure 2.3: Manipulator Model

A simple location graph is considered for now, which includes four different locations, which are initial location, object location, goal location and an obstacle location. The transition between different nodes can happen only when the action is "move" or "carry", and it will stay at current position when the action is "hold".

Definition 2.5. *Product Automaton The product of two finite automaton $\mathcal{F} = \mathcal{F}^{(i)} \times \mathcal{F}^{(j)} := (Q, Q_0, \alpha, \delta, \mathcal{F})$ is constructed as:*

(1) a set of states $Q = Q^{(i)} \times Q = Q^{(j)}$,

(2) a set of initial states $Q_0 = \{(q_i, q_j) \in Q : q_i \in Q_0^{(i)}, q_j \in Q_0^{(j)}\}$

(3) a set of Boolean formulas $\alpha = \alpha_i \wedge \alpha_j : \alpha_i \in \alpha^{(i)}, \alpha_j \in \alpha^{(j)}$

(4) transition conditions δ*:* $Q \times Q \to \alpha$ *defined below.*

The transition conditions of the product $\mathcal{F}$ need to capture the conditions of both NFAs $\mathcal{F}^{(i)}$, $\mathcal{F}^{(j)}$ and thus, are given by $\delta : ((q_i^s, q_j^s), (q_i^t, q_j^t)) \to \delta^{(i)}\ (q_i^s, q_i^t) \wedge \delta^{(j)}\ (q_j^s, q_j^t)$. A robotic agent is then described by a combination of its external environment and its internal state model. This combination can be done automatically from the product of two models given above.

Definition 2.6. *(Agent Model). The agent model is given as a product automaton*

$$\mathcal{A} = L \bigotimes R = (S_{\mathcal{A}}, A_{\mathcal{A}}, \Pi_{\mathcal{A}}, \wedge_{\mathcal{A}})$$

consisting of

(1) a set of states $S_{\mathcal{A}} = V_L \times S_R$ *combining location and internal state of the agent,*

(2) a set of actions $A_{\mathcal{A}} \subseteq S_{\mathcal{A}} \times S_{\mathcal{A}}$,

(3) a set of Boolean formulas $\alpha = \alpha_i \wedge \alpha_j : \alpha_i \in \alpha^{(i)}, \alpha_j \in \alpha^{(j)}$,

(4) transition conditions δ*:* $Q \times Q \to \alpha$ *defined below.*

Each node in the agent model is a position-action pair, and our goal is to find a sequence of position-action pair that satisfy the specifications.

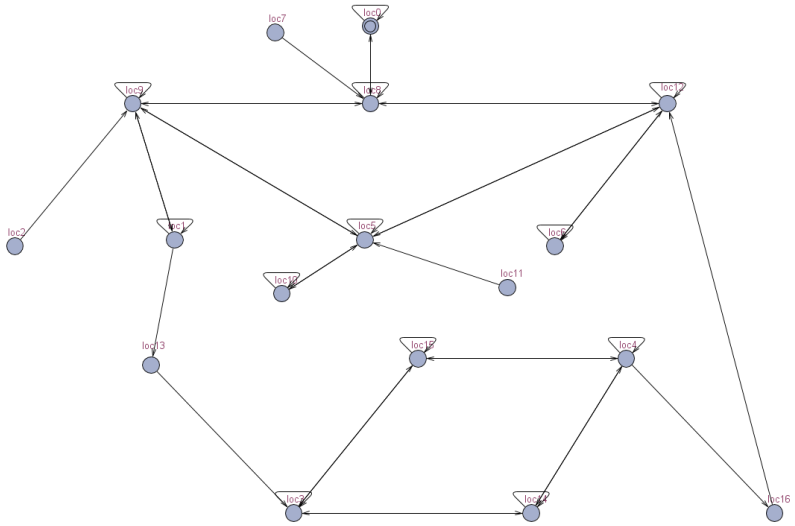


Figure 2.4: Agent Model, each node is a state-action pair. For example, node 0 represents (pos0, hold). For simplicity purpose, we assume the same guard condition for all edges.

Definition 2.7. *Timed Automata A timed automaton is a 4-tuple:*$A = (L, X, l_0; E)$

- *L is a finite set of locations*
- *X is a finite set of clocks*
- *$l_0 \in L$ is an initial location*
- *$E \subseteq L \times C(X) \times 2^X \times L$ is a set of edges, where C(X) are the clock constraints*

edge = (source location, clock constraint, set of clocks to be reset, target location)

2.3 From MITL to timed automata

Similar to what has been described in [5], in order to transform MITL specifications into timed automata, we first change every temporal logic operator into

a timed signal transducer, which is a temporal automaton that accepts input and generates output. The transformation of Until operator and timed Eventually operator is summarized in Figs. 2.5 and 2.6. The IOTA for timed eventually ($\Diamond_I$a) is decomposed into two automata, the generator generates predictions of the future outputs of the system, while the checker verifies that the generated outputs actually fit the inputs. Detailed derivations and verifications of the models can be found in [5]. The composition between them is achieved through the shared clock variables. Additional synchronization ('ch!') is added in our case to determine the final satisfaction condition for the control synthesis. A finite time trajectory satisfies the MITL, when the output signal of the generator automaton.

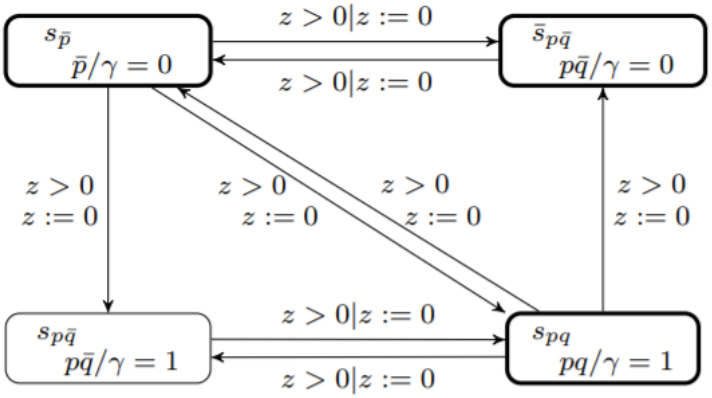
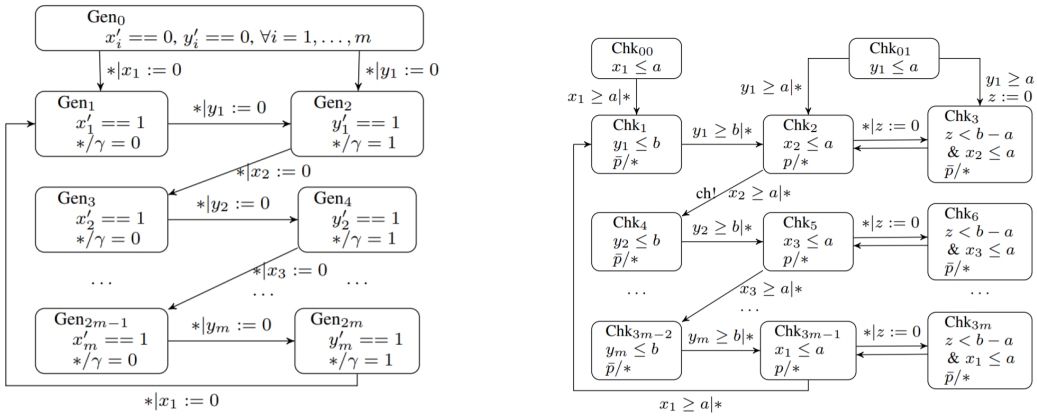


Figure 2.5: The timed automaton for p$\mathcal{U}$q. The inputs and outputs of the states are specified in the second line of each state. $p\bar{q}$ means the inputs are [1, 0] and $\bar{q}$ means the inputs can be [0, 1] or [0, 0], and $= 1$ means the output is 1. Transitions are specified in the format of guard|reset. In this case all the transitions have guard z ¿ 0 and reset clock z. All states in this automaton are Buchi accepting states except $s_{p\bar{q}}$. The Buchi accepting states are highlighted.

The overall framework is summarized as follows:

1. We construct the agent model by taking product of the environment model



(a) Timed Automaton for the generator

(b) Timed Automaton for the checker

Figure 2.6: The timed automaton for the generator part and the checker part of $\Diamond_I a$ for motion planning. 2m is the number of clocks required for the timed eventually ($\Diamond_I$) operator

and the robot model. We denote the agent model as $\mathcal{A}$

2. MITL formula is translated to IOTA, denoted as $\mathcal{M}$.

3. IOTA $\mathcal{M}$ is then taken product with the agent automaton model $\mathcal{A}$, we construct the timed automaton $\mathcal{G} = \mathcal{A} \bigotimes \mathcal{M}$

4. The resulting timed automata are then automatically transformed to an UPPAAL model with additional satisfaction condition verifier.

5. An optimal timed path is then synthesized using the UPPAAL verification tool.

2.4 Case Study

In this section we provide case study for two manipulation scenarios. The first one considers a pick-and-place manipulation task without specifying time constraint,

while the second one includes a time constraint in the specification.

Example 2.1. *(Example 1.) Given a task of robot grasping an object at a certain position and moving it to the goal position, while always avoiding the obstacle. The MTL formula could be expressed as follows:*

$$\phi_1 = (\neg\ pos\_goal\ \mathcal{U}\ pos\_object) \wedge (\neg\ pos\_goal\ \mathcal{U}\ grasp) \wedge (\Diamond\ pos\_goal\ ) \wedge (\Box \neg\ pos\_obs)$$

which requires do not go to goal position until the robot has visited object position and has grasped the object, and eventually move to the goal position. During the whole process, the robot should never collide with the obstacle. The sequence found by UPPAAL which satisfies the LTL formula is (See figure 5):

As shown in figure 2.7, the path found by UPPAAL is loc0 → loc6 → loc16 → loc13 → loc17 → loc42 → loc50 → loc35 → loc51_b , which corresponds to the following action sequence: (pos0, hold) → (pos0, move) → (pos3, move) → (pos3, hold) → (pos3, grasp) → (pos3, carry) → (pos0, carry) → (pos1b, carry) → (pos1b, release). By following the execution sequence, the robot is guaranteed to satisfy the given MITL formula. Here pos0 is the intial position, pos3 is the object position, pos1b is the goal position.

Example 2.2. *(Example 2.) We then consider another example which involves time constraints. We want the robot to pick up the object after 5 seconds but no later than 10 seconds, while avoiding the obstacle. The MITL formula could be written as follows:*

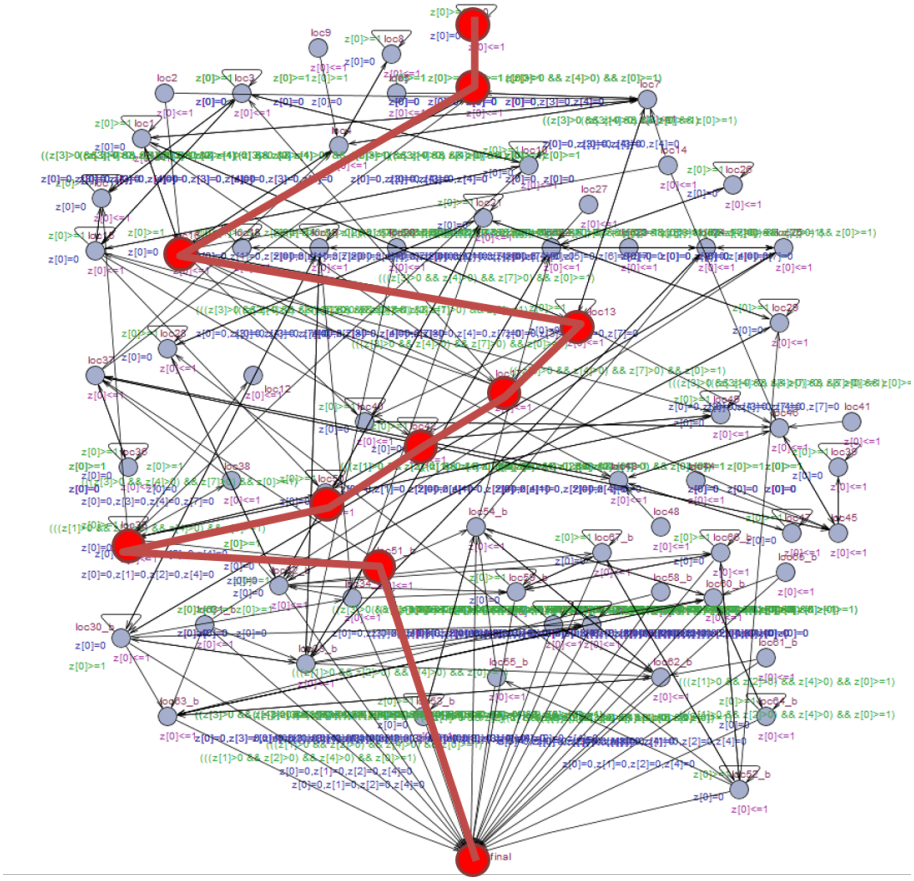


Figure 2.7: The Resulting timed automaton in UPPAAL of ϕ_1. The purple colored texts under the state names represent invariants. The green colored texts along the edges represent guard conditions, while the blue ones represent clock resets. The Buchi accepting states are represented by a subscript b in state names.

$$\phi_2 = (\neg\ grasp\ \mathcal{U}\ pos\_object) \wedge (\Diamond_{[5,10]}\ grasp) \wedge (\Box\ \neg\ pos\_obs\ )$$

Similar to Example 1, we use UPPAAL to find the execution sequence, and the resulting path is shown in Fig 2.8. The sequence found by UPPAAL is (loc117, loc139) $\rightarrow$ (loc117, loc133) $\rightarrow$ (loc125, loc133) $\rightarrow$ (loc126, loc133) $\rightarrow$ (loc118, loc133) $\rightarrow$ (loc118, loc133) $\rightarrow$ (loc16, loc133), which again corresponds to ((pos_init, hold), $t = 0$) $\rightarrow$ ((pos_init, hold), $t \in [0, 1]$) $\rightarrow$ ((pos_init, move), $t \in [1, 2]$) $\rightarrow$ ((pos_object, move), $t \in [2, 3]$) $\rightarrow$ ((pos_object, hold), $t \in [3, 4]$) $\rightarrow$ ((pos_object, hold), $t \in [4, 5]$)

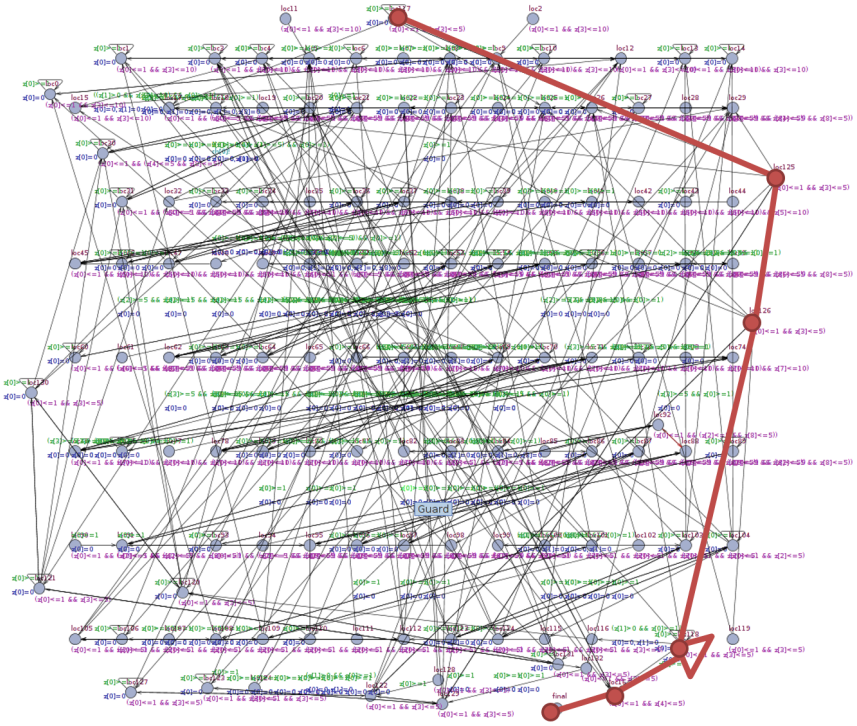


Figure 2.8: The Resulting timed automaton for checker part in UPPAAL of ϕ_2.

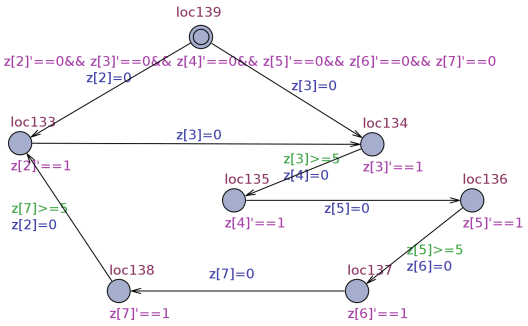


Figure 2.9: The Resulting timed automaton for generator part in UPPAAL of ϕ_2.

$\rightarrow$((pos_object, grasp), $t \in [5, 6]$) As can be seen from the result, the robot waits 1 time unit at position 3, such that it could meet the requirement of grasping the object after 5 seconds.

2.5 Chapter Summary

In this section, we have considered the automated generation of behaviors for a robotic manipulator while considering time constraints for both position changes (move from position A to position B) and for performing actions (grasping an object, releasing an object). We showed the execution sequence generated by UPPAAL for two different cases, and both of them satisfied the given specification. As the number of possible positions and number of possible actions increase, a major drawback is the state space explosion, which makes the planning practically unusable. The future direction we want to investigate is to see whether there are ways to aggregate "unimportant" states.

Chapter 3: Safety Monitor for Manipulation Tasks

Due to the uncertainty in the environment, the verification results obtained with respect to the system and environment models at design-time might not be transferable to the system behavior at run time. For autonomous systems operating in dynamic environments, safety of motion and collision avoidance are critical requirements. In this section, we will consider the run time monitoring for safety executions. We propose a two-phase process for our safety monitoring problem. In the design phase, we obtain an execution sequence for the robot which satisfies some desired specifications and has correctness guarantee. For the runtime phase, we model the robot as a hybrid system and we build a model monitor to check whether the execution sequence at runtime matches the desired execution sequence, and a safety monitor to check the runtime safety specifications of the system.

3.1 Related Work

Mitsch et al. use differential dynamic logic to verify safe obstacle avoidance for autonomous robotic ground vehicles with the dynamic window algorithm. Passive safety and passive friendly safety properties are proposed. Both properties are verified with respect to an environment which contains stationary as well as

moving obstacles. The autonomous vehicle is modeled as a hybrid system which describes the continuous physical motion of the robot as well as its discrete control choices. The paper in [20] presents the ModelPlex approach, which combines offline verification of CPS models with runtime validation in order to provide correctness guarantees for system executions at runtime. The method uses theorem proving with sound proof rules to synthesize three runtime monitors, i.e. model monitor, controller monitor and prediction monitor, from hybrid system models. The first monitor checks the system execution for deviations from the system model. The second monitor tests the current controller decisions of the system implementation for compliance with the system model, while the prediction monitor evaluates the worst-case safety impact of the current controller decisions with respect to the predictions of a bounded deviation plant model. LTL3, which is the 3-valued LTL, is designed for reasoning about LTL properties for finite executions and has been used for runtime verification [52]. LTL3 specifications could be transformed into a monitor automaton, where the transitions of the states are based on runtime sensory information. If the monitor automaton goes to the "bad" states, it implies that a fault is detected at runtime and the execution should be stopped. Runtime monitors can be used to sidestep the problem of verifying a (needfully) complex model of a robotic system. Instead of specifying and verifying the entire system, the properties that the system has to exhibit are extracted and specified as a monitor of the system. Runtime monitors can mitigate the problem of the reality gap (between a model and the real world) especially when used to compliment offline verification. Given that a robotic system is naturally cyber-physical, and therefore malfunctions

can have safety consequences, monitoring the system's behaviour at runtime can be key to safe operation.

3.2 Modeling of Hybrid System

Definition 3.1. *(Hybrid Automaton). A hybrid automaton is described by a tuple (Loc, Edge, $\sum$, X, Init, Inv, Flow, Jump) where the symbols have the following meanings.*

- *Loc is a finite set $l_1, l_2, ...l_n$ of (control) locations that represent control modes of a hybrid system.*
- *$\sum$ is a finite set of event names.*
- *Edge $\subseteq$ Loc $\times$ $\sum$ $\times$ Loc is a finite set of labelled edges that represent discrete changes of control mode in the hybrid system. Those changes are labelled by event names taken from the finite set of labels $\sum$.*
- *X is a finite set $\{x_1, x_2, ..., x_m\}$ of real-valued variables. We write $\dot{X}$ for the set of dotted variables $\{\dot{x_1}, ..., \dot{x_m}\}$ which are used to represent first derivatives of the variables during continuous evolutions (inside a mode),and we write X' for the primed variables $\{x_1', ..., x_m'\}$ that are used to represent updates at the conclusion of discrete changes (from one control mode to another).*
- *Init, Inv, Flow are functions that assign three predicates to each location. Init(l) is a predicate whose free variables are from X and which states the possible valuations for those variables when the hybrid system starts from location*

l. Inv(l) is a predicate whose free variables are from X and which constrains the possible valuations for those variables when the control of the hybrid system is in location l. Flow(l) is a predicate whose free variables are from $X \bigcup \dot{X}$ *and which states the possible continuous evolutions when the control of the hybrid system is in location l.*

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

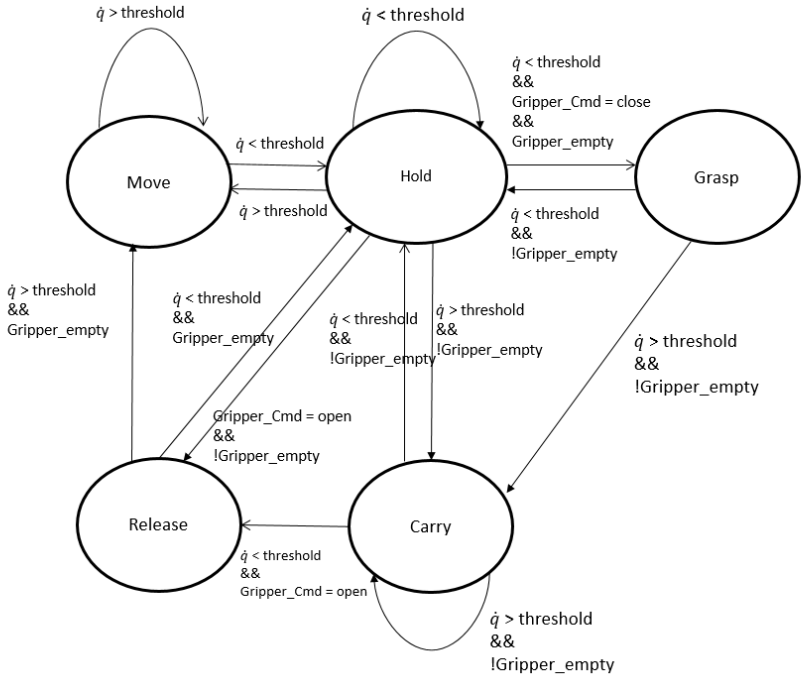


Figure 3.1: Hybrid Automaton Model for Manipulator

3.3 Runtime Monitoring

Runtime monitors can mitigate the problem of the reality gap (between a model and the real world) especially when used to compliment offline verification. Given that a robotic system is naturally cyber-physical, and therefore malfunctions can have safety consequences, monitoring the system's behavior at runtime can be key to safe operation. We propose a two-phase process for our safety monitoring problem. In the design phase, we obtain an execution sequence for the robot which satisfies some desired specifications and has correctness guarantee. For the runtime phase, we model the robot as a hybrid system and we build a model monitor to check whether the execution sequence at runtime matches the desired execution sequence, and a safety monitor to check the runtime safety specifications of the systemi.e, the *model monitor* and the *runtime safety monitor*.

3.3.1 Model Monitor Design

In order to design the model monitor, we construct a hybrid system model [4] for the manipulator, where the transition is based on the sensory information. The hybrid system model is demonstrated in Fig 3.1. Five different modes are considered, "hold" represents that the end-effector is staying at its current position, "move" represents that the end-effector is moving, "grasp" represents the action of closing the gripper, "carry" represents that the end-effector is moving while holding an object, and "release" represents the action of opening the gripper. The construction of the model monitor is then straight forward. In each state s_i, we test the previous

state s_{i-1} and check whether the transition follows the desired trace in design phase. If an error is detected, the execution should be stopped.

3.3.2 Safety Monitor Design

For designing the safety monitor, instead of specifying and verifying the entire system, the safety properties that the system has to exhibit are extracted and specified as a monitor of the system. We specify additional runtime safety specifications in LTL. LTL semantics is defined over infinite traces and a running program can only deliver a finite trace at a monitoring point. To formalize satisfaction of LTL properties for a finite trace at run time, in [52] the authors propose semantics for LTL3, where the evaluation of a formula ranges over three values {$\top$, $\bot$, ?} (denoted LTL3). The value '?' expresses the fact that it is not possible to decide on the satisfaction or violation of a property, given the current program finite trace. We denote the set of all finite words over $\sum$ by $\sum^*$ and the set of all infinite words by $\sum^\omega$. For a finite word u and a word w, we write u $\cdot$ w to denote their concatenation.

3.3.3 3-valued LTL

LTL semantics is defined over infinite traces and a running program can only deliver a finite trace at a monitoring point. To formalize satisfaction of LTL properties at run time, in [1], the authors propose semantics for LTL, where the evaluation of a formula ranges over three values {$\top$, $\bot$, ?} (denoted LTL3). The value '?' expresses the fact that it is not possible to decide on the satisfaction or violation of

a property, given the current program finite trace. We denote the set of all finite words over $\sum$ by $\sum^*$ and the set of all infinite words by $\sum^\omega$. For a finite word u and a word w, we write u · w to denote their concatenation.

Definition 3.2. *(LTL3 semantics). Let $\alpha \in \sum^*$ be a finite trace. The valuation of an LTL3 formula ϕ with respect to α, denoted by $[\alpha \models \phi]$, is defined as follows:*

$$[\alpha \models \phi] = \begin{cases} \top & \text{if} \quad \forall \omega \in \sum^\omega : \alpha \cdot \omega \models \phi \\ \bot & \text{if} \quad \forall \omega \in \sum^\omega : \alpha \cdot \omega \not\models \phi \\ ? & \text{otherwise} \end{cases}$$

Note that the syntax $[\alpha \models \phi]$ for Ltl3 semantics is defined over finite words as opposed to $\alpha \models \phi$ for Ltl semantics, which is defined over infinite words. For example, given a finite program trace $\sigma = a_0 a_1 ... a_n$, property $\Diamond p$ holds iff $a_i \models p$, for some i, $0 \leq i \leq n$. Otherwise, the property evaluates to ?. LTL3 specifications could be transformed into a monitor automaton [52], where the transitions between the states are based on runtime sensory information. If the monitor automaton goes to "bad" state, we should stop the execution. For example, in the pick-and-place example, we require that whenever the manipulator is grasping the object, the manipulator should not start moving until the force sensor confirms that the object is grasped firmly.

Definition 3.3. *(LTL Monitor Automaton). Let ϕ be an LTL formula over predicates Pred. The monitor automaton M^ϕ of ϕ is the unique deterministic finite-state*

automaton (DFA) $M^{\phi} = (Pred, Q, q_0, \delta, \lambda)$*, where* Q *is the set of states,* q_0 *is the initial state,* $\delta \subseteq Q \times Pred \times Q$ *is the transition relation, and* λ *is a function that maps each state in* Q *to a values in* $\{\top, \bot, ?\}$*, such that for any finite trace* $\alpha \in \sum^*$*:*

$$[\alpha \models \phi] = \lambda(\delta(q_0, \alpha))$$

For example, after the robot received a command of grasping (closing the gripper), we require the coordinate of all joint positions should remain the same until the force sensor has confirmed that the grasping is finished (for example, Force $>$ 1N).

$$\phi_m = \Box(grasp \rightarrow (v = 0)\,\mathcal{U}\,(Force > 1))$$

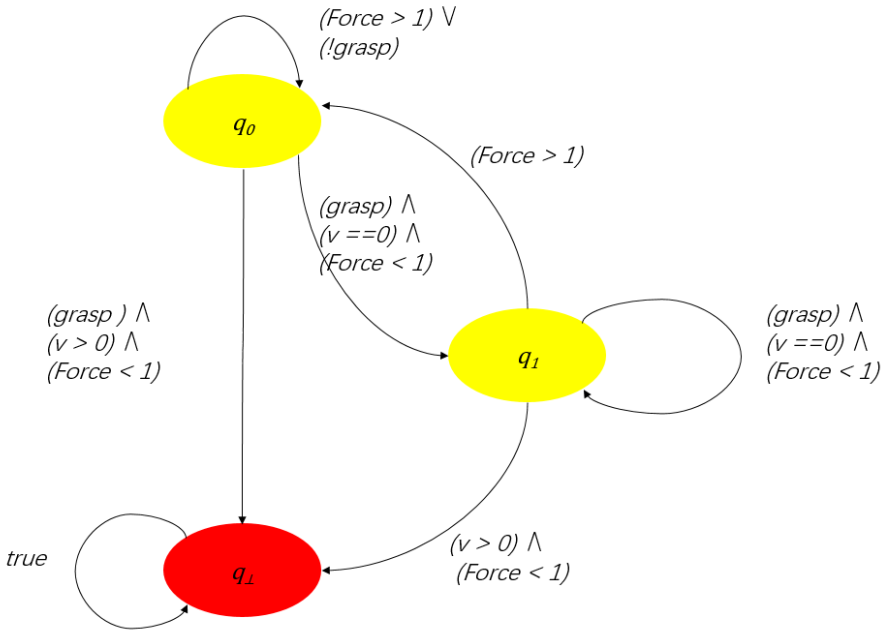


Figure 3.2: The corresponding monitor automaton for ϕ_m

3.4 Implementation

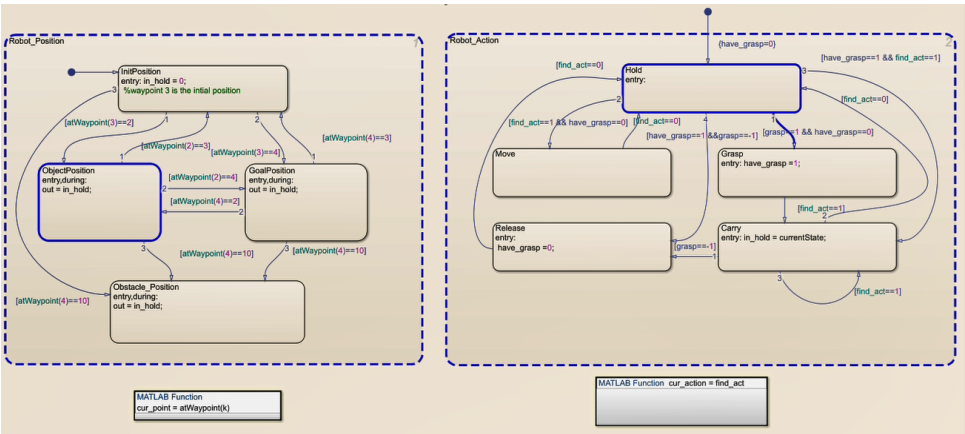


Figure 3.3: Model monitor built in Stateflow. Current local state is highlighted in blue boundary. Figure shows the sample state during the runtime monitoring. The robot is currently at state (pos_obj, hold).

In order to perform the runtime analysis, the behavior described in the robot and obstacle models had to be transferred in executable source code. The transformation from UPPAAL models to executable source code was done using automata-based programming. We built the robot model in Matlab/Simulink and the runtime monitors with Stateflow. The model monitor is shown in figure 3.3, where the states correspond to the position-action pair, and the guard condition for the transition depends on the sensory information. For example, the robot position state changes from current state to a new state pos_new when $\|X_{ee} - pos\_news\| < \delta$, where δ is the tolerance threshold and X_{ee} is the end-effector position calculated using the forward kinematics equation; the robot action state changes from "hold" to "move" when the velocity of the end-effector is not 0 and also when the force of the gripper is 0 (i.e, not holding anything).

The safety monitor is shown in figure 3.2, where we require the coordinate of

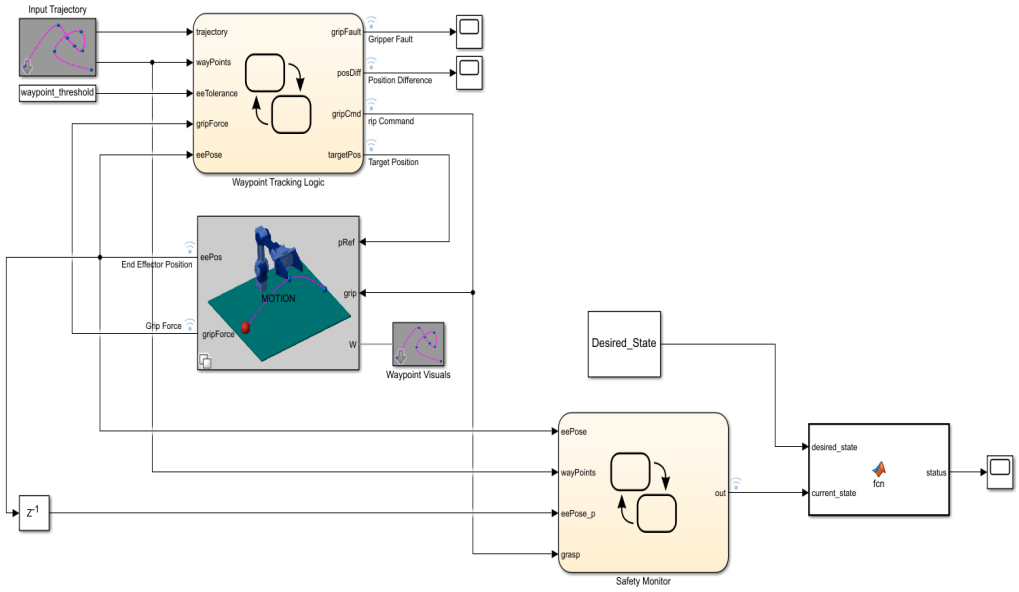


Figure 3.4: Complete Simulink Model considering robot dynamics

all joint positions should remain the same until the force sensor has confirmed that the object is indeed grasped by the gripper (for example, Force > 1N). If the model monitor detects a violation, then the verified properties in the design phase may no longer hold for the system. If the safety monitor detects a "bad" state, it means the LTL3 specifications for the safety requirement is violated. Provably safe fail-safe actions should be taken in these situations.

We conducted 5 different experiments which includes 1 correct execution and 4 faulty executions. The task specification for the robot is given as $\phi_1 = (\neg\, pos\_goal\, \mathcal{U}\, pos\_object) \wedge (\neg\, pos\_goal\, \mathcal{U}\, grasp) \wedge (\Diamond\, pos\_goal\,) \wedge (\Box \neg\, pos\_obs)$ and the additional safety requirement is $\phi_m = \Box(grasp \rightarrow (v = 0)\, \mathcal{U}\, (Force > 1))$ as discussed before. The faulty executions include the robot receives "grasp" command at wrong positions, the robot receives "release" command at wrong positions, the robot moves to a wrong position, and the robot starts moving before the grasping

finishes. All 4 faulty executions are successfully detected, where the first three cases are detected by the model monitor and the last case is detected by the safety monitor.

3.5 Chapter Summary

In this section, we first modeled the manipulator as a hybrid system, then we dicussed the design of two runtime monitors. A model monitor is designed to monitor the correctness of the task execution, and a safety monitor is designed based on LTL3 specifications to guarantee the safety during the execution. In the following chapters, we will discuss possible ways for self-correction while error has been detected in the runtime.

Chapter 4: MITL based Reinforcement Learning with Runtime Monitoring and Self-Correction

In this chapter, we present a modular Q-learning framework to deal with the robot task planning, runtime monitoring and self-correction problem. The task is specified using metric interval temporal logic (MITL) with finite time constraints. We first construct a runtime monitor automaton using three-valued LTL (LTL3), and a sub-task MITL monitor is constructed by decomposing and augmenting the monitor automaton. During the learning phase, a modular Q-learning approach is proposed such that each module could learn different sub-tasks. During runtime, the sub-task MITL monitors could monitor the execution and guide the agent for possible self-correction if an error occurs. Our experiments show that under our framework, the robot is able to learn a feasible execution sequence that satisfies the given MITL specifications under finite time constraints. When the runtime environment becomes different than the learning environment and the original action will violate the specifications, the robotic agent is able to self-correct and accomplish the task if it is still possible.

4.1 Related Work

Guiding a mobile robot to accomplish the desired tasks safely and quickly is one of the essential topics in robotic planning. To adapt to unknown environments, learning ability is important for mobile robots. Reinforcement learning has widely been applied to robot path planning [70], and in recent years, researchers start to associate reinforcement learning with temporal logic constraints. Linear temporal logic (LTL) allows one to specify more complicated mission tasks that are hard to express and to achieve by conventional methods in classical reinforcement learning [49]. Reinforcement learning with linear temporal logic (LTL) specifications has been considered in several recent studies, such as [57] [65] [66] [72] [73]. The basic idea is to translate the temporal logic specification into an automata-based structure, model the agent as a markov decision process (MDP) and construct a product MDP for assigning the rewards. Time behavior is a most important issue for the autonomous systems of interest, and it is critical for many robotic tasks. The above methods with LTL, however, are not capable of providing the framework needed for the autonomous systems to plan under finite time constraints. Metric Interval Temporal Logic (MITL) and signal temporal logic (STL) allows finite time constraints to be added to the specifications. Reinforment learning with MITL and STL are considered in [74] and [68], respectively. However, both works make the assumption that the runtime environment is the same as the training environment and only a single objective is considered. In practice, this is not always the case. During runtime execution, there may be additional obstacles in the environment;

and a robotic investigation task could be specified as: "Investigate either position a or position b, and return the information to position c", where the task could be accomplished by multiple ways. In this chapter, we propose a sub-task monitor design based on the temporal logic specifications, and enables the re-planning ability for the robotic agent when the runtime environment is different from the training environment. Traditional Q-learning algorithms will have trouble executing the task correctly if the environment has changed since the Q values are trained and stored offline. We propose a modular Q-learning method which is able to monitor the runtime task execution, and try to self-correct if the original plan fails. A number of two-valued semantics for LTL on finite traces have been proposed [75] [76] [77]. In monitoring a property, there arise at least three different situations: in the first case, the property is satisfied after a finite number of steps, independently of the future continuation; second, the property is shown to evaluate to false for every possible continuation, and third, the finite, already observed prefix still allows different continuations leading to either satisfaction or falsification. However, all the two-valued logic must evaluate to true or false prematurely since it cannot reflect the third and inconclusive case properly. LTL3, which is the 3-valued LTL, is designed for reasoning about LTL properties for finite executions and has been used for runtime verification [52]. Runtime monitor design for robotic agents and distributed system has been discussed in [62] [1]. We propose to first convert the untimed temporal logic specifications into a LTL3 monitor automaton, and then leverage the unique deterministic property to design our sub-task MITL monitor automaton by decomposing and augmenting the LTL3 monitor automaton.

The contributions of this chapter are as follows. First, we propose the design of the sub-task MITL monitor automaton by decomposing and augmenting the LTL3 monitor automaton, and use it to guide the modular Q-learning process, where each module could learn a different sub-task from the specification. Second, we propose to define reward functions for each learning module based on the sub-task monitor progression. Finally, we use the sub-task MITL monitors to guide possible self-corrections when re-planning is needed during runtime.

4.2 Preliminaries

Definition 4.1. *An atomic proposition is a statement about the system variables (x) that is either* ***True*** *($\top$) or* ***False*** *($\bot$) for some given values of the state variables.*

Definition 4.2. *(Linear Temporal Logic) The syntax of LTL formulas are defined according to the following grammar rules:*

$$\phi ::= T \,|\, \pi \,|\, \neg\phi_1 \,|\, \phi_1 \wedge \phi_2 \,|\, \bigcirc\phi_1 \,|\, \phi_1 \,\mathcal{U}\, \phi_2 \,|\, \phi_1 \,\mathcal{R}\, \phi_2 \tag{4.1}$$

With LTL formulas ϕ_1, ϕ_2, propositions $\pi \in \Pi$, and the Boolean constant T "true ". The syntax includes the Boolean operators $\neg$ "not" and $\wedge$ "and", as well as the temporal operator $\bigcirc$ "next", $\mathcal{U}$ "until", and "release". The temporal operators "or": $\phi_1 \vee \phi_2$, "implies": $\phi_1 \rightarrow \phi_2$, "eventually": $\Diamond\phi_1$, "always": $\Box\phi_1$ can be represented using the grammar described in the definition.

To formalize satisfaction of LTL properties at run time, in [52], the authors

propose semantics for LTL3, where the evaluation of a formula ranges over three values $\{\top, \bot, ?\}$ (denoted LTL3). The value '?' expresses the fact that it is not possible to decide on the satisfaction or violation of a property, given the current program finite trace. We denote the set of all finite words over $\sum$ by $\sum^*$ and the set of all infinite words by $\sum^\omega$. For a finite word u and a word w, we write u · w to denote their concatenation.

Definition 4.3. *(LTL3 semantics). Let $\alpha \in \sum^*$ be a finite trace. The valuation of an LTL3 formula ϕ with respect to α, denoted by $[\alpha \models \phi]$, is defined as follows:*

$$[\alpha \models \phi] = \begin{cases} \top & \text{if} \quad \forall \omega \in \sum^\omega : \alpha \cdot \omega \models \phi \\ \bot & \text{if} \quad \forall \omega \in \sum^\omega : \alpha \cdot \omega \not\models \phi \\ ? & \text{otherwise} \end{cases}$$

Note that the syntax $[\alpha \models \phi]$ for LTL3 semantics is defined over finite words as opposed to $\alpha \models \phi$ for LTL semantics, which is defined over infinite words. For example, given a finite program trace $\sigma = a_0 a_1 ... a_n$, property $\Diamond p$ holds if and only if $a_i \models p$, for some i, $0 \leq i \leq n$. Otherwise, the property evaluates to ?. LTL3 specifications could be transformed into a monitor automaton [52] to monitor the runtime execution. In the following context, we denote the states that are evaluated to "$\top$" as "good states" (green), states that are evaluated to "$\bot$" as "bad states" (red), and states that are evaluated to "?" as "neutral states" (yellow).

Definition 4.4. *(LTL3 Monitor Automaton). Let ϕ be an LTL formula over atomic propositions AP. The monitor automaton M^ϕ of ϕ is the unique deterministic finite-*

state automaton (DFA) $M^\phi = (AP, Q^\phi, q_0, \delta, \lambda)$*, where* Q^ϕ *is the set of states,* q_0^ϕ *is the initial state,* $\delta \subseteq Q^\phi \times AP \times Q^\phi$ *is the transition relation, and* λ *is a function that maps each state in* Q^ϕ *to a values in* $\{\top, \bot, ?\}$*, such that for any finite trace* $\alpha \in \sum^*$*:*

$$[\alpha \models \phi] = \lambda(\delta(q_0^\phi, \alpha))$$

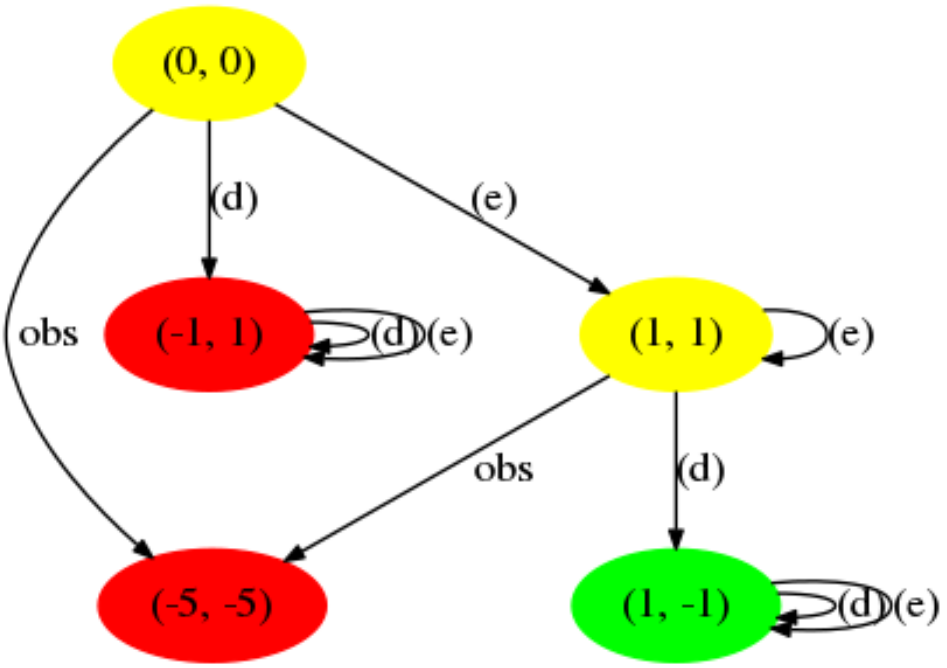


Figure 4.1: The monitor automaton for property $\phi = (\neg d \,\mathcal{U}\, e) \wedge (\Diamond d)$, the yellow states are the "neutral states", the green state is the "good state" and the red state is the "bad state"

For example, Fig 4.1 shows the monitor automaton for property $\phi = (\neg d \,\mathcal{U}\, e) \wedge (\Diamond e)$, where $\lambda((0,0)) = \lambda((1,1)) = ?$, $\lambda((-1,1)) = \bot$ and $\lambda((1,-1)) = \mathrm{T}$. The specification requires position d is not visited until position e is visited, and eventually d needs to be visited. Notice that state $(-1,1)$ and $(1,-1)$ is a final state with no outgoing transition to other states. This is because once verdicts $\top$ or $\bot$

are reached, according to Definition 4.2.3, they cannot change.

Remark 4.1. *However, there remain many properties which are non-monitorable: consider for example the specification* $\phi = \Box(a \rightarrow \Diamond b)$*, which requires "once position a has been visited, position b will always be visited eventually." No finite word is a good or bad prefix for* ϕ *and therefore, this property is always evaluated to an "?" state. [64] discusses a more detailed evaluation of such an inconclusive state for LTL3 by defining "presumably false" and "presumably true" states, where such kind of evaluation is not possible using LTL.*

Definition 4.5. *(Metric Interval Temporal Logic) The syntax of MITL formulas are defined according to the following grammar rules:*

$$\phi ::= \top \,|\, \pi \,|\, \neg\phi_1 \,|\, \phi_1 \wedge \phi_2 \,|\, \bigcirc\phi_1 \,|\, \phi_1 \,\mathcal{U}_I\, \phi_2 \,| \tag{4.2}$$

where e I $\subseteq [0, \infty]$. $\mathcal{U}_I$ symbolizes the timed Until operator. Sometimes we will represent $\mathcal{U}_{[0,\infty]}$ by $\mathcal{U}$. Other Boolean and temporal operators such as conjunction ($\vee$), eventually within I ($\Diamond_I$), always on I ($\Box_I$) etc. can be represented using the grammar described in the definition. For example, we can express time constrained eventually operator $\Diamond_I\phi \equiv \top\, \mathcal{U}_I\phi$ and so on.

Definition 4.6. *The semantics of any MTL formula* ϕ *is recursively defined over a trajectory* (ξ, t) *as:*

- $(\xi, t) \models \pi$ *iff* (ξ, t) *satisfies* π *at time* t
- $(\xi, t) \models \neg\phi_1$ *iff* (ξ, t) *does not satisfiy* π *at time* t

- $(\xi, t) \models \phi_1 \wedge \phi_2$ *iff* $(\xi, t) \models \phi_1$ *and* $(\xi, t) \models \phi_2$
- $(\xi, t) \models \circ\phi_1$ *iff* $(\xi, t+1) \models \phi_1$
- $(\xi, t) \models \phi_1 \mathcal{U}_I \phi_2$ *iff* $\exists\ s \in I$ *such that* $(\xi, t+s) \models \phi_2$ *and* $\forall s' \leq s$ *it holds that* $(\xi, t+s') \models \phi_1$

Definition 4.7. *(Timed Automata) A timed automaton is a 4-tuple:*$A = (L, X, l_0; E)$

- L *is a finite set of locations*
- X *is a finite set of clocks*
- $l_0 \in L$ *is an initial location*
- $E \subseteq L \times C(X) \times 2^X \times L$ *is a set of edges, where C(X) are the clock constraints*

Definition 4.8. *(Robot Model). The robot model* $\mathcal{R} = (S, s_0, A, T, AP, L)$ *is a tuple over a finite set of states* S *where* A *is a finite set of actions.* $T \subseteq S \times A \times S$ *is the transition relation, and* s_0 *is the initial state.* AP *is a finite set of atomic propositions and a labeling function* $L : S \to 2^{AP}$ *assigns to each state* $s \in S$ *a set of atomic propositions* $L(s) \subseteq 2^{AP}$*. Assume that the set of available actions at state* s *is* $A_s \subseteq A$*. We use* $s \xrightarrow{a'} s'$ *to denote a transition from state* $s \in S$ *to state* $s' \in S$ *by action* $a' \in A_s$*.*

In this section, we consider a grid-like environment where the state of the robot is its location and the robot is able to move in four directions with different velocities, i.e, the set of actions for the robot is $A = \{left_1, right_1, up_1, down_1, left_2, right_2, up_2, down_2\}$, where $left_1$ means move one unit left, and up_2 means move two units up, etc. We

could use the labeling function to assign the atomic proposition to each state. For example, we could label the locations of interest using a, b, c, etc, and all locations inside the black circles as "obs". Then the specification $\phi_m = (\neg a \mathcal{U} b) \wedge \Diamond a \wedge \Box \neg obs$ means "visit position b first (marked as yellow circle) before reaching position a (marked as red circle), and eventually reaching position a while avoiding obstacles at all times."

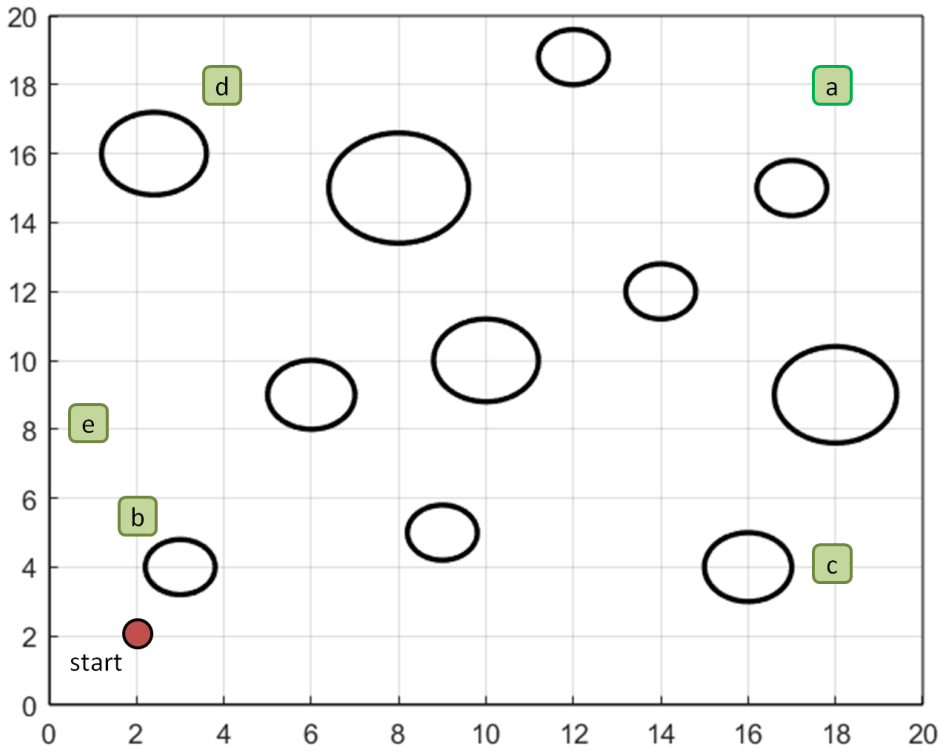


Figure 4.2: Task Planning Workspace. The red circle is the starting point, a, b, c, d, e are the locations of interests. *obs* is the set of obstacle locations, and the obstacles are marked in black.

4.3 Monitor Guided Modular Q-learning

In this section, we discuss our modular reinforcement learning process guided by the sub-task monitor. We first discuss how to generate the sub-task monitor

and guide our modular Q-learning process, and then we discuss how the sub-task monitor automaton could be used to guide the runtime correction process.

4.3.1 Classical Q-learning Algorithm

The Q-learning algorithm [21] uses the notation $Q(S, a)$ to represent the value function. Here, (S, x) is called the state action pair. S is the state of a given agent and a is the action we take at that state. $\hat{q}^n$ can be viewed as the observation and can be calculated as follows:

$$\hat{q}^n = R(S^n, a^n) + \gamma \max_a Q^{n-1}(S^{n+1}, a). \tag{4.3}$$

We then update our value function using the observations:

$$Q^n(S^n, x^n) = (1 - \alpha_{n-1})Q^{n-1}(S^n, a^n) + \alpha_{n-1}\hat{q}^n \tag{4.4}$$

Here S^n is the current state and a^n is the action we take at this state. $R(S^n, a^n)$ is the immediate reward of a state action pair, α_n is the step size at each iteration, γ is the forgetting factor, and S^{n+1} is the next state given the current state and the action. The convergence of Q-learning is guaranteed as long as we visit every state-action pair infinitely often and choose a suitable step size [21]. One drawback of classical Q-learning is that once the training process is finished, the Q values for every state and action pair become fixed. If the environment becomes different than the training environment, the trained policy may no longer be correct.

4.3.2 Modular Q-Learning Algorithm

In practice, it is quite often we do not only have a single objective for the robotic task. Karlsson [69] developed approaches to the problem of multiple-goal reinforcement learning, where a separate learning module is created for each component MDP. Following a similar idea, given a monitor automaton, we define a sub-task monitor as follows:

Definition 4.9. *(Sub-task LTL Monitor). Given a LTL3 monitor automaton* $M^{\phi} = (AP, Q^{\phi}, q_0, \delta, \lambda)$*, a sub-task monitor* M^{ϕ}_{LTLsub} *is defined as a transition system starting from* q_0*, ending in a state* q' *such that* $\lambda(q') = \top$*, and has the following property: (1) each edge in* M^{ϕ} *has been visited at most once (2) transition to good or neutral state accepts exactly one atomic proposition, all other atomic propositions leads to bad states. (3) only one good state in a sub-task monitor*

For example, Fig 4.5 shows the resulting LTL3 monitor automaton M^{ϕ} for $\phi = ((\neg d \,\mathcal{U}\, e) \rightarrow \Diamond_{[5,10]} d) \vee (\neg a \,\mathcal{U}\, (b \rightarrow \Diamond_{[10,20]} c) \wedge \Diamond a) \wedge (\Box \neg obs)$ while neglecting the time constraints. The monitor automaton is generated automatically using ltl3tool[1]. We could then build sub-task LTL monitors according to Definition 4.9. We then generate the sub-task MITL monitor by augmenting time constraints according to the atomic propositions and its corresponding *root task*.

Definition 4.10. *(Root Task) An MITL formula with structure* $\phi = (\phi_1 \vee \phi_2 \cdots \vee \phi_m) \wedge \phi_s$ *is equivalent to* $\phi = (\phi_1 \wedge \phi_s) \vee (\phi_2 \wedge \phi_s) \cdots \vee (\phi_m \wedge \phi_s)$*. We denote each*

[1]The tool is available on `http://ltl3tools.sourceforge.net/`

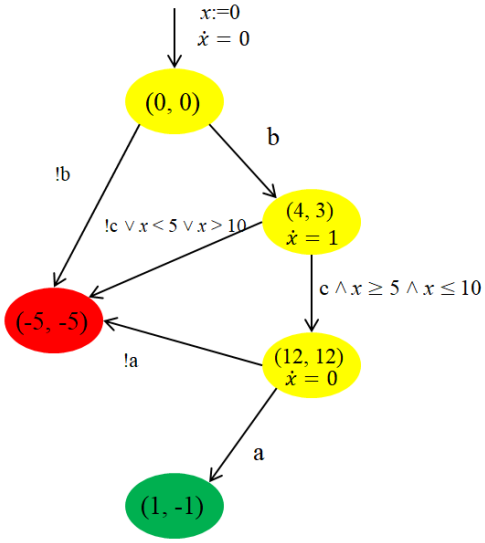
$F_i = (\phi_i \wedge \phi_s)$ as the root task of the specification. The MITL formula is satisfied by satisfying any of the root task.

In this chapter, we assume that it is not possible to accomplish two different root tasks with the same execution sequence, i.e, for a finite sequence α, it is not possible that $[\alpha \models F_i]$ and $[\alpha \models F_j]$ for $i \neq j$. This is often true in practice, since otherwise there is no need to set different objectives for the robotic agent. Under this assumption, we could always associate the sub-task LTL monitor with its corresponding root task.

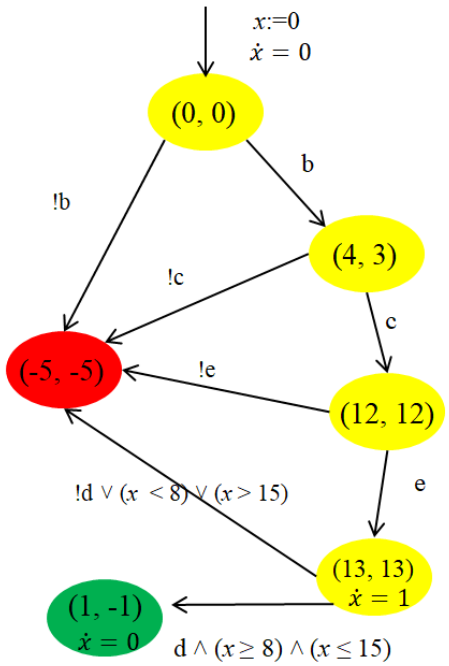
Definition 4.11. *(Sub-task MITL Monitor). Given a sub-task LTL monitor M^{ϕ}_{LTLsub} and its corresponding root task F_i, let Ω denote the set of atomic propositions for root task F_i. If F_i contains temporal operators such as $[\cdot]\,\mathcal{U}_{[t_1,t_2]}\,\rho$,$\Diamond_{[t_1,t_2]}\rho$, or $\Box_{[t_1,t_2]}$ for $\rho \in \Omega$, a sub-task MITL monitor M^{ϕ}_{sub} can be constructed by adding a clock constraint $t_1 \leq x \leq t_2$ to the transition that accepts exactly ρ on M^{ϕ}_{LTLsub}.*

Fig 4.3 shows two sub-task MITL monitors for the MITL specification ϕ. Note that the unique construction of the sub-task MITL monitor is possible due to the unique deterministic property of LTL3 monitor automaton [52], i.e, given a finite trace α, the transition $\delta(q^{\phi}_0, \alpha)$ is deterministic. This is not the case with the traditional LTL to Büchi automaton construction, where the Büchi automaton is non-deterministic [71].

Remark 4.2. *In this work we consider each atomic proposition is only attached to at most one clock constraints, i.e, specifications such as $\Diamond_{[t_1,t_2]}\Box_{[t_3,t_4]}\rho$ are not considered in this chapter.*



(a) MITL monitor $M_{sub}^{\phi,3}$



(b) MITL monitor $M_{sub}^{\phi,4}$

Figure 4.3: Two sub-task MITL monitors for M^{ϕ}. Note that in (a), clock constraint is added to the transition with condition c, but in (b) no clock constraint is added to the transition with condition c. This is because the root task is different for (a) and (b). x is the clock variable. $\dot{x} = 0$ means the clock is deactivated and x will not change. $\dot{x} = 1$ means the clock is active and x will increase by one after taking an action.

We perform the reinforcement learning in the extended state space, $S_i^{ext} = S \times Q_i^{\phi} \times V^{\phi}$, where $i \in \{1, 2, \cdots, N\}$ is the index for the sub-task MITL monitor, N is the total number of sub-task monitors for the given specification, and is equal to the number of distinct *simple paths* (each edge in M^{ϕ} can be visited at most once in a simple path) from q_0 to q', S is the state for the workspace, Q^{ϕ} is the sub-task MITL monitor state, and V^{ϕ} is the set of clock valuations for the clocks in X. The clock is initially deactivated by setting $\dot{x} = 0$, and will only activate once it reaches a state with invariant $\dot{x} = 1$. Figure 4.4 shows an illustrative example for the extend state space while considering the specification $\phi = (\Diamond_{[2,3]} A) \wedge (\Box \neg O)$.

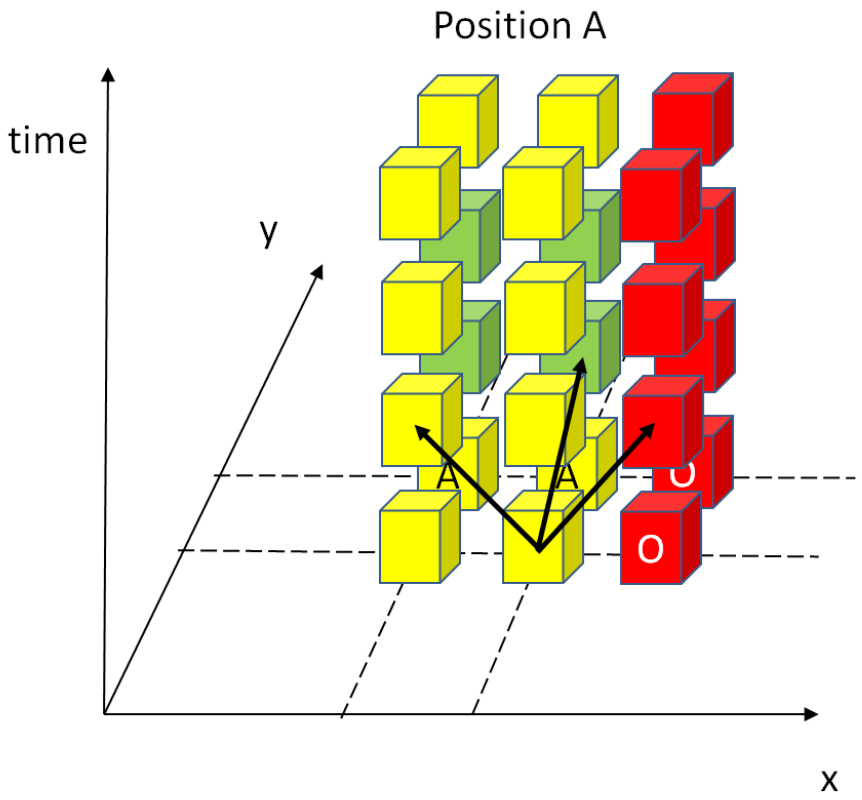


Figure 4.4: Q-learning in the extended space for specification $\phi = (\Diamond_{[2,3]} A) \wedge (\Box \neg O)$. The state represents location A turns green only within the time interval [2,3].

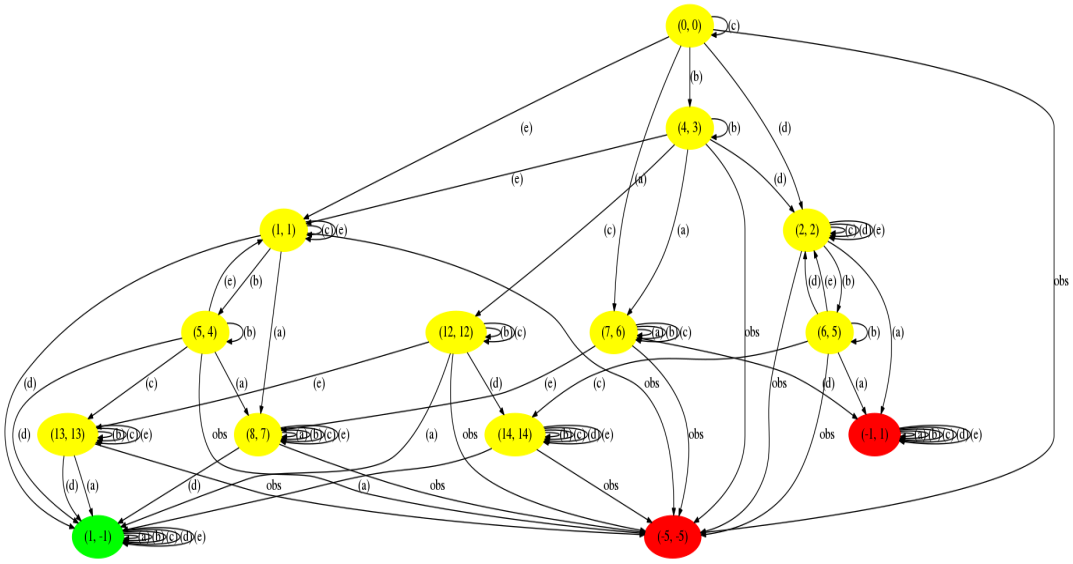


Figure 4.5: LTL3 monitor automaton M^{ϕ} for $\phi = ((\neg d \,\mathcal{U}\, e) \rightarrow \Diamond_{[8,15]} d) \vee (\neg a \,\mathcal{U}\, (b \rightarrow \Diamond_{[5,10]} c) \wedge \Diamond a) \wedge (\Box \neg obs)$. Transitions with multiple labels such as $a \wedge b$ are removed since it is impossible to be at position a and b at the same time.

4.3.3 Reward Function

Consider a task which requires a sequential order of visiting different positions with time constraints, it is not enough to define our reward function only based on the positions, since this could not capture neither the sequential requirement nor the time constraints. In this work we propose to define our reward function over the sub-task MITL monitor automaton based on the progression of satisfying the temporal logic specification. Since each sub-task monitor is trained separately, the reward function will also be different for each learning process.

Definition 4.12. *(Sub-task Monitor Progression). Let M^{ϕ}_{sub} be a sub-task monitor, $Q^{\phi}_{\top}$ be the "good" state , $Q^{\phi}_{\bot}$ be the "bad" state, and q^{ϕ} is the current state. We define the sub-task monitor progression using a metric d. d is the shortest distance*

from q^ϕ to $Q^\phi_\top$. Note that $Q^\phi_\top$ has a d value of 0, and any state in $Q^\phi_\bot$ has a d value of infinity.

With the definition of monitor progression, we could define the reward function as follows. The reward function observes the current state s and action a and also observes the subsequent state s' and gives the agent a scalar reward according to the following rule:

$$R_{s\otimes,a} = \begin{cases} r_p & if \quad [\alpha \models \phi] \neq \bot \quad and \quad d(s) > d(s') \\ r_n & if \quad [\alpha \models \phi] = \bot \quad and \quad d(s') = \infty \\ r_s = 0 & if \quad [\alpha \models \phi] \neq \bot \quad and \quad d(s) = d(s') \end{cases} \tag{4.5}$$

Assumption 4.1. *(Safe exploration). We assume the robotic agent is able to recognize the surrounding locations, i.e, the locations it could reach with one single action from current position. During the learning process, the agent $\mathcal{A}$ receives an action command a at state s, it will only execute the command if the next state $s' \notin Q^\phi_\bot$. The agent will receive a penalty for the action but the action will not be executed if $s' \in Q^\phi_\bot$.*

The reward function will give a positive reward r_p if the next state has a better progression, i.e. smaller d value. It will give a negative reward r_n if the next state is a "bad state". It will give a neutral reward $r_s = 0$ if the progression stays the same. Note that it is possible to get a positive reward even if the monitor automaton is at a neutral state, as long as it progress to a smaller "d" value with the current action.

A separate Q-table is created for each sub-task MITL monitor.

$$\hat{q}^n = R(S_i^{ext,n}, a^n) + \gamma \max_a Q_i^{n-1}(S_i^{ext,(n+1)}, a). \tag{4.6}$$

We then update our value function using the observations:

$$Q_i^n(S_i^{ext,n}, a^n) = (1 - \alpha_{n-1})Q_i^{n-1}(S_i^{ext,n}, a^n) + \alpha_{n-1}\hat{q}^n \tag{4.7}$$

The complete procedure of our modular Q-learning framework is summarized in Algorithm 1.

Algorithm 1 Monitor Guided Reinforcement Algorithm

1: Transform MITL specification ϕ into a LTL3 monitor automaton
2: Construct all sub-task MITL monitors according to Definition slowromancapiii@.1 - Definition slowromancapiii@.3
3: Select a sub-task Monitor $M_{sub}^{\phi,i}$, $\forall i \in \{1, 2, \cdots, N\}$
4: Construct the extended state S_i^{ext}
5: Initialize Q values for each of the module as 0
6: **Start episode**
7: Initialize the agent at a random location

- Choose exploit or explore according to ϵ-greedy exploration
- Receive reward based on the sub-task monitor progression (4.5)
- Update Q value based on equations (4.6) and (4.7)

8: **End episode if the agent reaches a "good state"**
9: Return to step 3 and repeat the process

4.4 Runtime monitoring and self-correction

During runtime, only one of the sub-task MITL monitors $M_{sub}^{\phi,i}$ will be selected to guide the execution initially. Since there may be multiple ways of accomplishing the task, one problem is which sub-task monitor we should use during runtime. We could have different criteria on selecting our sub-task monitor during runtime execution. One possible choice could be selecting the shortest distance in the workspace such that the specification is satisfied. Note that this does not necessary correspond to the smallest number of states in the sub-task monitor automaton. Another possible choice could be selecting the highest accumulating rewards, which normally corresponds to a path with less obstacles around. Another important problem is, when the runtime environment becomes different than the training environment (e.g, some of the states in the workspace becomes unreachable), the initial execution sequence may not be able to accomplish the task. We propose our self-correction method given as below.

Proposition 4.1. *(Runtime Self-Correction). Let M_S^ϕ denote the set of all sub-task MITL monitors for the task ϕ. Assume the runtime execution is initally guided by sub-task monitor $M_{sub}^{\phi,i} \in M_S^\phi$ and $\alpha \in \sum^*$ be the finite timed trace until current state s. The sub-task monitor will switch if at current state s, the agent receives an action command a' such that $s \xrightarrow{a'} s' \in Q_\perp^\phi$, and there exist another sub-task monitor $M_{sub}^{\phi,j}$ such that $\lambda(\delta(q_0^\phi, \alpha)) \neq \perp$, then the sub-task monitor will be switched to $M_{sub}^{\phi,j}$.*

During runtime execution, while only one module is guiding the execution

initially, other MITL monitors are running until they transit to a bad state. When a sub-task MITL monitor enters the bad state, the monitor will be turned off and infers that the corresponding sub-task is impossible to complete. Due to our safe exploration Assumption 4.1, if the agent receives a command a at current state s that leads to a bad state, i.e, $s' \in Q_{\perp}^{\phi}$, the command will not be executed, however, the current MITL monitor will be turned off and the agent will check if it is still able to accomplish the task by switching to another MITL monitor.

4.5 Case Studies

In this section, we provide case studies for our monitor guided reinforcement learning algorithm. The task is to find a feasible execution sequence that satisfies the MITL specifications, starting from the initial position. The initial position of the robot is marked as the red circle, and the black circles represent the obstacles. The size of the environment is 20 by 20. (x,y) is the position of the robot in cartesian space, x and y can take any integer values between 1 and 20. The robot could move in four directions $left, right, top, down$ with speed 1 or 2. We present the results of two different scenarios: (1) the runtime environment is the same as the training environment (2) the runtime environment is different than the training environment, where a rectangle obstacle is added.

The task we consider in the case study is: "The robot can accomplish the task by achieving either one of the objectives: (1) do not visit position d until information is collected at position e, then once position e has been visited, eventually return to

position d between 8 and 15 time units to report the information; (2) do not visit position a until information is collected at position b, and after visiting b, the robot has to immediately visit position c between 5 and 10 time units without visiting other locations, and eventually return to position a to report the information." This could be written using MITL as:

$$\phi = ((\neg d \,\mathcal{U}\, e) \rightarrow \Diamond_{[8,15]} d) \vee (\neg a \,\mathcal{U}\, (b \rightarrow \Diamond_{[5,10]} c) \wedge \Diamond a) \wedge (\Box \neg obs)$$

The resulting LTL3 monitor automaton M^{ϕ} is shown in Fig 4.5. Four sub-task monitors of interest are presented in Table 1. Fig 4.6 shows the resulting path guided by the sub-task MITL monitor $M^{\phi,3}_{sub} = \{b, c, a\}$ when the runtime environment is the same as the training environment. The robot is able to satisfy the MITL specification by visiting b,c and then a. Note that after visiting position b, the robot picks the maximum speed (move 2 steps a time whenever it is possible) to reach position c, since otherwise position c could not be reached within 10 time units after visiting b. In contrast, the robot selects speed randomly moving from c to a, since there is no time constraint for visiting a.

Fig 4.7 shows the resulting path when the runtime environment becomes different from the training environment. In this case, position a becomes unreachable. Initially the robot still selects sub-task MITL monitor $M^{\phi,3}_{sub} = \{b, c, a\}$, however, the original execution sequence will lead to an obstacle while trying to reach position a. Following Proposition slowromancapiii@.1, the agent switches the MITL monitor to

sub-task monitors	atomic propositions	corresponding states in M^{ϕ}
$M^{\phi,1}_{sub}$	$\{e, d\}$	$(0,0) \rightarrow (1,1) \rightarrow (1,-1)$
$M^{\phi,2}_{sub}$	$\{b, e, d\}$	$(0,0) \rightarrow (4,3) \rightarrow (1,1) \rightarrow$
$M^{\phi,3}_{sub}$	$\{b, c, a\}$	$(0,0) \rightarrow (4,3) \rightarrow (12,12) \rightarrow (1,-1)$
$M^{\phi,4}_{sub}$	$\{b, c, e, d\}$	$(0,0) \rightarrow (4,3) \rightarrow (12,12) \rightarrow (13,13) \rightarrow (1,-1)$

Table 4.1: Four sub-task monitors for ϕ_1, each could guide the robot to satisfy the specification

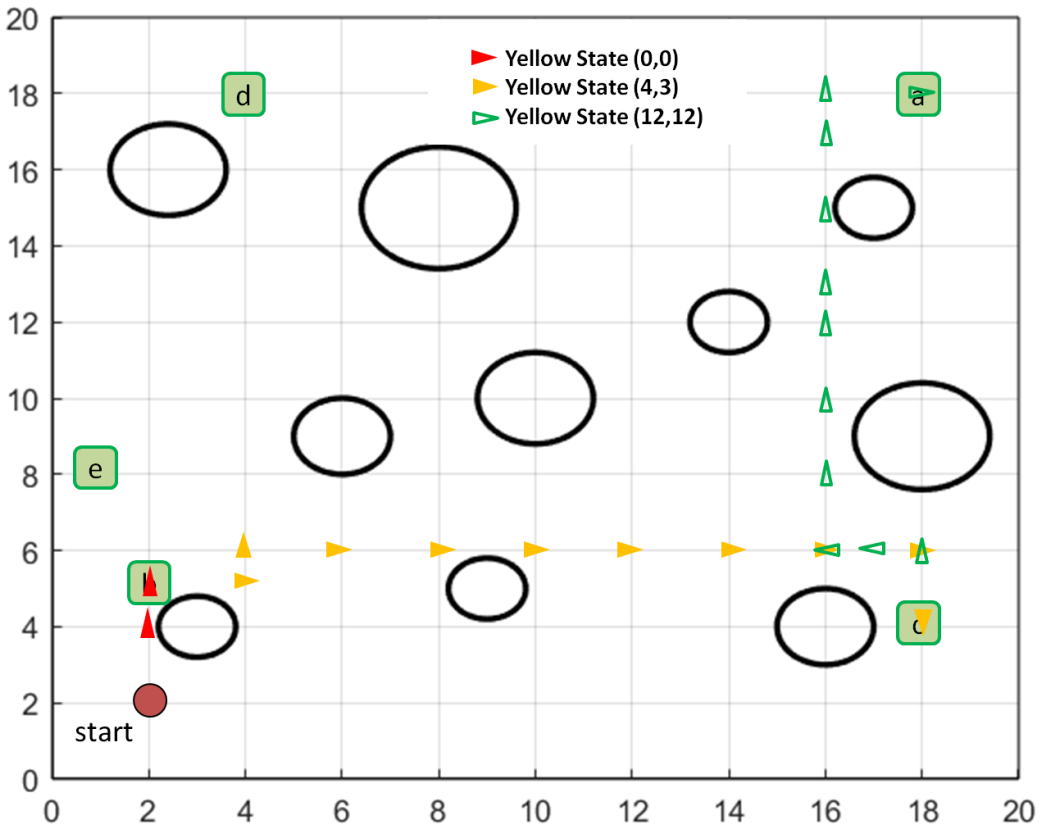


Figure 4.6: Resulting path by following the sub-task monitor $M^{\phi,3}_{sub} = \{b, c, a\}$. Runtime environment remains the same as the learning environment.

$M_{sub}^{\phi,4} = \{b, c, e, d\}$. Unlike moving from b to c, the robot does not move full speed (move 2 steps a time whenever it is possible) from position e to position d. This is due to the time constraint that position d has to be visited between 8 and 15 time units after visiting e.

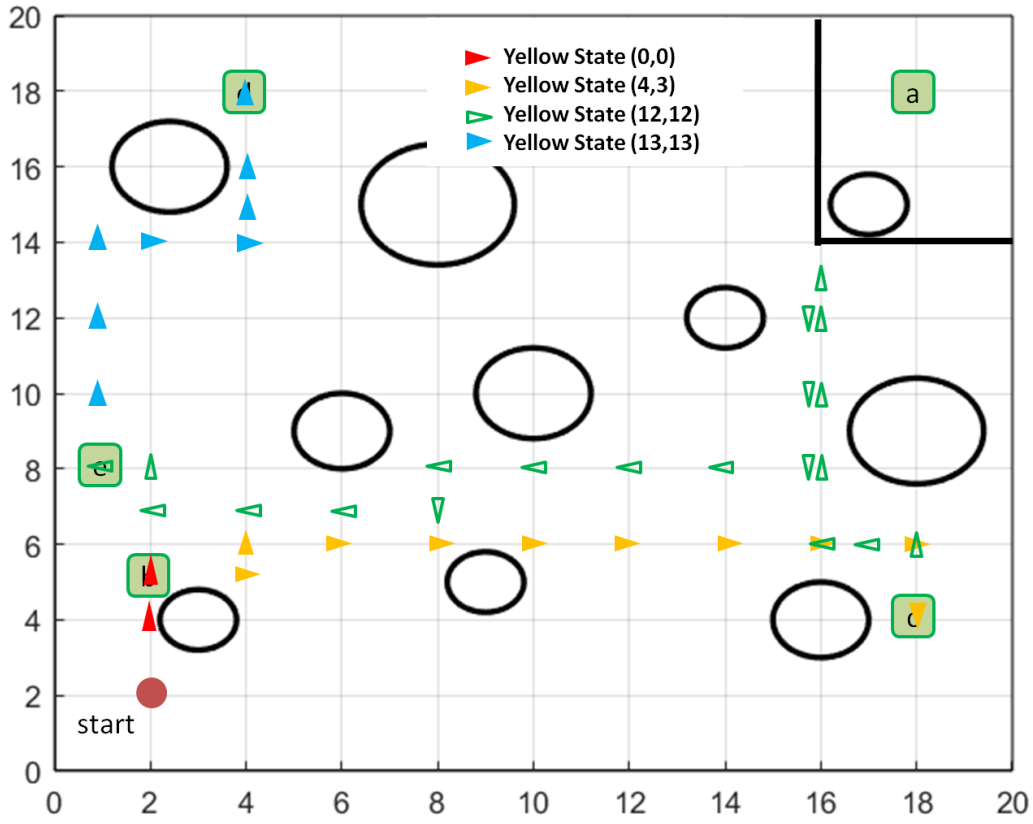


Figure 4.7: Resulting path by following the sub-task monitor $M_{sub}^{\phi,3} = \{b, c, a\}$ initially and switch to $M_{sub}^{\phi,4} = \{b, c, e, d\}$ when the robot figures out "a" is not reachable.

4.6 Chapter Summary

In this chapter, we have proposed a modular reinforcement learning framework based on the design of sub-task monitors. Given a specification with time

constraints, we construct the LTL3 monitor automaton, and a set of sub-task MITL monitors is then generated based on the LTL3 monitor automaton. A modular Q-learning framework is used and we defined the reward function for each module based on the sub-task monitor progression. During runtime execution, a sub-task monitor is selected initially depending on the desired criterion to guide the robot execution. When the task can not be accomplished, the robot is able to switch to another sub-task monitor and tries to self-correct. Simulation results show that with the framework we proposed, the robot is able to learn a feasible execution sequence that satisfies the given MITL specification under finite time constraints. When the runtime environment becomes different than the learning environment and the original action will violate the specification, the robotic agent is able to self-correct and accomplish the task if it is still possible.

Chapter 5: Optimization-based Motion Planning for Robotic Systems with Space and Time Tolerances

In previous chapters, we have considered automata-based approach for robot planning. However, this relies on abstraction of the environment and many low-level details are ignored. In this chapter we present an optimization-based approach for robot planning, monitoring and self-correction problems under signal temporal logic specifications (STL), where the exact control commands could be generated and both space and time tolererances are considered. The STL specifications are translated into mixed-integer linear constraints, and we generate the reference trajectory by solving a mixed-integer-linear-programming (MILP) to maximize the overall space and time tolerances. During runtime execution, a prediction module is constantly evaluating the robustness degree of the predicted trajectory, and a self-correction module based on event-triggered model predictive control (MPC) has been designed to predict and correct possible future violations of the specifications. Simulation results show that with our approach, the robotic agent is able to generate a path that satisfies the STL specifications while maximizing space and time tolerances, and able to make corrections when there are possible violations of the specifications during runtime execution.

5.1 Related Work

Motion and task planning for autonomous robotic agents is important in many real, physical world applications. Robotic agents have been deployed for agriculture research, surveillance, and search and rescue operations. In recent years, a new approach to the task planning problem for robotic agents has evolved by formulating system specifications in temporal logics [12] [14] [16]. Linear temporal logic (LTL) allows one to specify more complicated mission tasks that are hard to express and to achieve by conventional methods [78].

However, LTL specifications do not emphasize finite time constraints. For real applications, a robot might be required to perform a specific task within a certain time bound, rather than at some arbitrary time in the future. Metric temporal logic (MTL) [3] and Signal Temporal Logic (STL) [82] have been introduced for motion planning with bounded time constraints. Task planning with bounded time constraints has been investigated in [26] and [79] by solving a MILP problem. An automata based approach for task planning with MTL specifications has been considered in [1].

STL allows the specification of properties of dense-time and real-valued signals. One main advantage of STL is the quantitative semantics which, in addition to the yes/no answer to the satisfaction question, provide a real number that grades the quality of the satisfaction or violation. This robustness information could be useful in motion planning, since we normally want the robotic agents to stay far away from the obstacles, and also to stay close to the center of the locations of interest. Motion

planning problems with STL have been investigated in [84] [85], and authors in [79] and [83] have used STL for control synthesis together with Model Predictive Control (MPC).

For autonomous systems operating in dynamic environments, the safety of motion and time requirements for the task are critical. Due to the uncertainty in the environment, the planning results obtained with respect to the system and environment models at design-time might not be transferable to the system behavior at run time. Therefore, allowing both space and time tolerances in the planning phase, and the ability of runtime monitoring and self-correction are essential. Monitoring problems for STL have been discussed in [80], [81] and [82]. [79] discusses MPC for signal temporal logic specifications, but their approach requires solving the MILP problem at each time step, and is not able to address the time robustness issue. [83] considers the motion planning problem using STL and introduces the Discrete Average Space Robustness to maximize the space robustness. However, in their work time robustness is not considered in the planning phase and the execution is assumed to be perfect. To address these issues, in this work we divide the problem into the following two parts: (1) offline control synthesis, (2) online monitoring and self-correction. We first generate a path that considers both space and time robustness, and then we design a prediction module and event-triggered MPC module for the monitoring and self-correction.

The contributions of this chapter are as follows. First, we transform the robot planning problem under signal temporal logic specifications into a mixed-integer linear programming problem, while considering both the time tolerances and space

tolerances. To the best of our knowledge, this is the first work that considers both space and time tolerances in the planning phase. Second, we have designed a monitoring and self-correction framework for the runtime execution. A predicted trajectory is generated at each time step, and we propose an event-triggered model predictive control framework such that the robot is able to make self-corrections when there is predicted error during runtime execution.

5.2 Preliminaries

Definition 5.1. An atomic proposition is a statement about the system variables (x) that is either True($\top$) or False($\bot$) for some given values of the state variables [13].

Definition 5.2. (STL semantics) The syntax of Metric Temporal Logic (MTL) formulas are defined according to the following grammar rules:

$$\phi ::= \mathrm{T} \,|\, \pi \,|\, \neg\phi_1 \,|\, \phi_1 \wedge \phi_2 \,|\, \Box_I \phi_1 \,|\, \phi_1 \,\mathcal{U}_I\, \phi_2 \,| \tag{5.1}$$

where I $\subseteq$ [0, ∞]. $\mathcal{U}_I$ symbolizes the timed Until operator. Sometimes we will represent $\mathcal{U}_{[0,\infty]}$ by $\mathcal{U}$. Other Boolean and temporal operators such as conjunction ($\vee$), eventually within I ($\Diamond_I$) etc. can be represented using the grammar described in the definition. For example, we can express time constrained eventually operator $\Diamond_I \phi \equiv \mathrm{T}\, \mathcal{U}_I \phi$ and so on.

For any signal s, let s_t denote the value of s at time t and let $(s,t) = s_t s_{t+1} s_{t+2} \cdots$ be the part of the signal that is a sequence of $s_t{}'$ for $t' \in [t, \infty)$.

Accordingly, the Boolean semantics of STL is recursively defined as follows:

- $(s,t) \models (f(s) < d) \Leftrightarrow f(s_t) < d$,
- $(s,t) \models \neg(f(s) < d) \Leftrightarrow \neg((s,t) \models (f(s) < d))$,
- $(s,t) \models \phi_1 \wedge \phi_2 \Leftrightarrow (s,t) \models \phi_1 \quad \text{and} \quad (s,t) \models \phi_2$,
- $(s,t) \models \phi_1 \vee \phi_2 \Leftrightarrow (s,t) \models \phi_1 \quad \text{or} \quad (s,t) \models \phi_2$,
- $(s,t) \models \Box_{[a,b]}\phi \Leftrightarrow (s,t') \models \phi \quad \forall t' \in [t+a, t+b]$,
- $(s,t) \models \Diamond_{[a,b]}\phi \Leftrightarrow \exists t' \in [t+a, t+b] \quad \text{s.t} \quad (s,t') \models \phi$.

Thus, the expression $\phi_1 \, \mathcal{U}_I \, \phi_2$ means that ϕ_2 will be true within time interval I and until ϕ_2 becomes true ϕ_1 must be true.

Definition 5.3. (Space Robustness) STL is endowed with a metric called robustness degree [82] (also called "degree of satisfaction") that quantifies how well a given signal s satisfies a given formula ϕ. The robustness degree is calculated recursively according to the quantitative semantic:

- $r(s, (f(s) < d), t) = d - f(s_t)$,
- $r(s, \neg(f(s) < d), t) = -r(s, (f(s) < d), t)$,
- $r(s, \phi_1 \wedge \phi_2, t) = \min(r(s, \phi_1, t), r(s, \phi_2, t))$,
- $r(s, \phi_1 \vee \phi_2, t) = \max(r(s, \phi_1, t), r(s, \phi_2, t))$,

- $r(s, \Diamond_{[a,b]}\phi, t) = \max_{t' \in [t+a, t+b]} r(s, \phi, t')$,
- $r(s, \Box_{[a,b]}\phi, t) = \min_{t' \in [t+a, t+b]} r(s, \phi, t')$,

Definition 5.4. (Time Robustness) The left and right time robustness of an STL formula ϕ with respect to a trace s at time t are defined as follows

- $\theta^{-}(s, f(s), t) = \max(d \geq 0 \text{s.t.} \forall t' \in [t-d, t], (s,t) \models \phi \Leftrightarrow (s,t') \models \phi)$
- $\theta^{+}(s, f(s), t) = \max(d \geq 0 \text{s.t.} \forall t' \in [t, t+d], (s,t) \models \phi \Leftrightarrow (s,t') \models \phi)$

The time robustness indicates how much the signal could be shifted to the left (right) such that the specification is still satisfied.

Assumption 5.1. (Double Integrator dynamics) The dynamics of the robot is assumed to be given by the following model:

$$X(k+1) = A \cdot X(k) + B \cdot U(k) \tag{5.2}$$

$$A = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} (\Delta t^2/2) & 0 \\ 0 & (\Delta t^2/2) \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \tag{5.3}$$

$$\mathbf{X}(t) = \begin{bmatrix} x(t) & y(t) & \dot{x(t)} & \dot{y(t)} \end{bmatrix}^T \tag{5.4}$$

where $x(t), y(t)$ are the cartesian position of the robotic agent, and $\dot{x(t)}, \dot{y(t)}$ are the velocities on each direction respectively. Let us denote the trajectory of the system starting at t_0 with initial condition x_0 and input $u(t)$ as $\mathbf{X}_{t0}^{x0,u} = \{\mathbf{X}(s)|s \geq t_0, \mathbf{X}(t+1) = f(t, \mathbf{X}(t), u(t)), \mathbf{X}(t_0) = x_0\}$. For brevity, we will use $\mathbf{X}_{t_0}$ instead of $\mathbf{X}_{t0}^{x0,u}$ whenever we do not need the explicit information about $u(t)$ and x_0. Satisfaction of a temporal specification ϕ by a trajectory $\mathbf{X}_{t_0}$ will be denoted as $\mathbf{X}_{t_0} \models \phi$.

5.3 Maximum Space-Time Tolerances Planning

Space and time tolerances are important for the planning problem. With large space tolerances, the robot has a higher chance to satisfy the temporal logic specifications when the trajectory deviates from the planning path. With large time tolerances, the robot could handle the situation when the execution is slower or faster than the plan. Therefore, it is important to take both space and time tolerances into consideration. As shown in Fig 5.1, if space and time tolerances are not taken into considerations, all three signals are considered as satisfying $\Diamond_{[a,b]}(x > 0)$ from $t = 0$ at the same degree. However, it is clear that the space tolerance of ω_2 is small (the specification will be violated if we disturb x a little) and the time tolerance for ω_3 is small (the specification will be violated if we shift the signal a little to the right).

The planning problem considered in this chapter is to determine the optimal trajectories such that the given temporal logic specifications are satisfied, and maximizing the space and time tolerances at the same time. The optimization problem

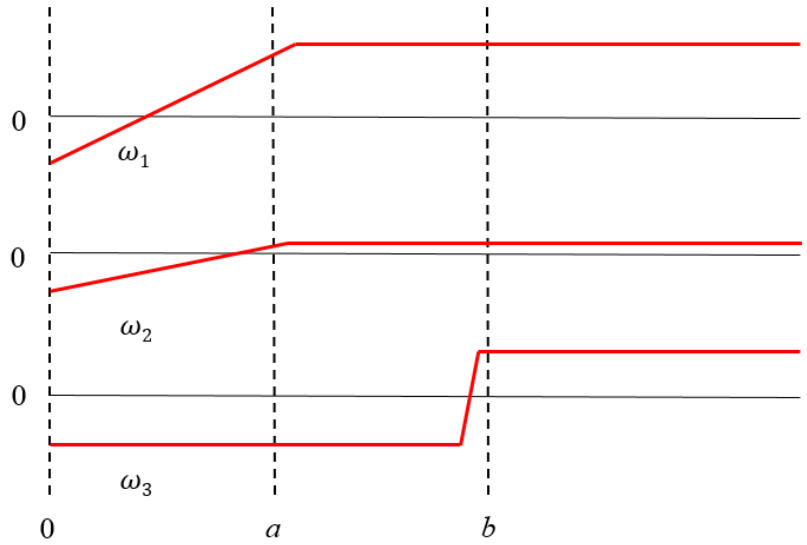


Figure 5.1: Limitations of the point-wise quantitative semantics: signals ω_1, ω_2 and ω_3 are considered as satisfying $\Diamond_{[a,b]}(x > 0)$ from $t = 0$ at the same degree.

could be formulated as follows:

$$\begin{aligned} \max_{\mathbf{X}(t),u(t)} \quad & \lambda_1 r^{\phi}_{time}(\mathbf{X}_{t_0}) + \lambda_2 r^{\phi}_{space}(\mathbf{X}_{t_0}) \\ \text{subject to} \quad & \mathbf{X}(t+1) = f(\mathbf{X}(t), u(t)) \\ & u_{min} \leq u(t) \leq u_{max} \\ & \mathbf{X}_{t_0} \models \phi \end{aligned} \tag{5.5}$$

- $r^{\phi}_{time}(\mathbf{X}_{t_0})$ and $r^{\phi}_{space}(\mathbf{X}_{t_0})$ are the time and space tolerances for the trajectory $\mathbf{X}_{t_0}$ over specification ϕ.

- The system dynamics are given in equation (5.2), and the control inputs are bounded to $[u_{min}, u_{max}]$.

- $\mathbf{X}_{t_0} \models \phi$ is the constraint that the STL specifications are satisfied.

The temporal logic constraints ϕ is transformed into linear constraints and will be described in details in the next section. The objective function we consider here is to maximize the overall space and time tolerances, which is

$$r^{\phi}(\mathbf{X}_{t_0}) = \lambda_1 \cdot r^{\phi}_{time}(\mathbf{X}_{t_0}) + \lambda_2 \cdot r^{\phi}_{space}(\mathbf{X}_{t_0}) \tag{5.6}$$

where λ_1 and λ_2 are the weight coefficients, and $\lambda_1 + \lambda_2 = 1$. $r_{space}(\mathbf{X}_{t_0})$ is the space tolerance, which is defined similarly as the space robustness in Definition 5.3. $r_{time}(\mathbf{X}_{t_0})$ is the time tolerance, and our goal is to generate a trajectory that is robust to time shifting. Therefore, we propose to extend the definition of time robustness as follows:

For eventually ($\Diamond_{[t_a,t_b]}A$) operator, the time tolerance is defined as follows:

$$r^{\phi}_{time}(\mathbf{X}_{t_0}) = \sum_{t=t_a}^{t_b} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{(t-\frac{t_a+t_b}{2})^2/2\sigma^2} \cdot P_t^A \tag{5.7}$$

where σ is a user-defined parameter indicating the standard deviation. Basically, we want to maximize the time that the robotic agent is staying within the locations of interests, and preferably in the middle of the allowed time interval. P_t^A is a binary variable and the value is 1 when the robot is within location A at time t and it is 0 otherwise. More detail discussion on how to construct P_t^A will be described in the next section.

For always ($\square_{[t_a,t_b]}A$) operator, the time tolerance is defined as follows:

$$r^{\phi}_{time}(\mathbf{X}_{t_0}) = \sum_{t=t_a-\tau}^{t_b} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{(t-t_a)^2/2\sigma^2} \cdot P^A_t + \sum_{t=t_b}^{t_b+\tau} \cdot \frac{1}{\sigma\sqrt{2\pi}} e^{(t-t_b)^2/2\sigma^2} \cdot P^A_t \tag{5.8}$$

where τ is a user-defined parameter and we want the robot to also satisfy the specification before t_a and after t_b. Note that the satisfaction of the specifications is already enforced by the constraint $\mathbf{X}_{t_0} \models \phi$, and we are maximizing the space and time tolerances in the objective function based on that.

5.4 Mixed Integer Linear Programming

In this section, we demonstrate our approach to translate a time-bounded temporal logic formula (constraint $\mathbf{X}_{t_0} \models \phi$ in equation (5.3)) to mixed integer linear constraints on state variables and inputs. We first need to express the temporal constraint that $x(t)$ lies within the area of interest $\mathcal{P}$ at time t in order to express the requirement for the motion. Any convex polygon can be represented as an intersection of several halfspaces. If the area of interest has a non-convex shape, we could always decompose the polygon to convex ones and link them using disjunction operators. A halfspace is expressed by a set of points, $\mathcal{H} = \{x : h_i^T x \leq k_i\}$. Thus, $x(t) \in \mathcal{P}$ is equivalent to $x(t) \in \cap_{i=1}^{n} \mathcal{H}(i)$. In order to translate the temporal constraints with location atomic propositions into mixed integer convex (linear) constraints, we use a similar method as discussed in [26].

In a polygonal environment, atomic propositions (AP), $p \in \Pi$, can be related to states of the system using disjunction and conjunction of halfspaces. In other words, the relationship between measured outputs such as the location of the robotic agent and the halfspaces defines the propositions used in the temporal logic. Consider the convex polygon case and let $z_i^t \in \{0, 1\}$ be the binary variables associated with halfspaces $\{x(t) : h_i^T x \leq k_i\}$ at time $t = 0, \cdots, N$. We enforce the following constraint $z_i^t = 1$ if and only if $h_i^T x \leq k_i$ by adding the convex (linear) constraints,

$$\begin{aligned} h_i^T x &\leq k_i + M(1 - z_i^t) \\ h_i^T x &\geq k_i - M z_i^t + \epsilon \end{aligned} \tag{5.9}$$

where M is a large positive number and ϵ is a small positive number. If we denote $P_t^{\mathcal{P}} = \wedge_{i=1}^n z_i^t$, then $P_t^{\mathcal{P}} = 1$ if and only if $x(t) \in \mathcal{P}$ at time t, and 0 otherwise. Therefore, $P_t^{\mathcal{P}}$ is the binary variable that indicates whether the robotic agent lie in area $\mathcal{P}$ at time t. Let p and q denote labels for some location in the environment. The following Boolean operators, such as $\neg$, $\wedge$, $\vee$, can be translated into linear constraints. For $t \in \{0, 1, ..., N\}$, we denote the variables associated with formula ϕ made up with propositions $p \in \Pi$ at time t as P_t^{ϕ}. The next subsection will discuss the construction of P_t^{ϕ} for different temporal logic specifications.

5.4.1 MTL to Mixed Integer Linear Constraints

Let p and q denote labels for some locations in the environment.

- The negation operation, $\phi = \neg p$ is modeled as

$$P_t^{\phi} = 1 - P_t^{p} \tag{5.10}$$

- The conjunction operation, $\phi = \wedge_{i=1}^{m} p_i$ is modeled as

$$\begin{aligned} &P_t^{\phi} \leq P_t^{p_i}, i = 1, \cdots m, \\ &P_t^{\phi} \geq 1 - m + \sum_{i=1}^{m} P_t^{p_i} \end{aligned} \tag{5.11}$$

- The disjunction operation, $\phi = \vee_{i=1}^{m} p_i$ is modeled as

$$P_t^{\phi} \geq P_t^{p_i}, i = 1, \cdots m; \quad P_t^{\phi} \leq \sum_{i=1}^{m} P_t^{p_i} \tag{5.12}$$

Similarly, the temporal operators can be modeled using linear constraints as well. Let $t \in \{0, 1, \cdots, N - t_2\}$, where $[t_1, t_2]$ is the time interval used in the MTL.

- Eventually: $\phi = \Diamond_{[t_1,t_2]} p$ is equivalent to

$$\begin{aligned} &P_t^{\phi} \geq P_{\tau}^{p}, \tau \in \{t + t_1, \cdots, t + t_2\} \\ &P_t^{\phi} \leq \sum_{\tau=t+t_1}^{t+t_2} P_{\tau}^{p} \end{aligned} \tag{5.13}$$

- Always: $\phi = \Box_{[t_1,t_2]} p$ is equivalent to

$$\begin{aligned} P_t^\phi &\leq P_\tau^p, \tau \in \{t+t_1, \cdots, t+t_2\} \\ P_t^\phi &\geq \sum_{\tau=t+t_1}^{t+t_2} P_\tau^p - (t_2 - t_1) \end{aligned} \tag{5.14}$$

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

$$\begin{aligned} & a_{tj} \leq P_j^q, j \in \{t+t_1, \cdots, t+t_2\} \\ & a_{tj} \leq P_k^p, k \in \{t, \cdots, j-1\}, j \in \{t+t_1, \cdots, t+t_2\} \\ & a_{tj} \geq P_j^q + \sum_{k=t}^{j-1} P_k^p - (j-t), j \in \{t+t_1, \cdots, t+t_2\} \\ & P_t^\phi \leq \sum_{j=t+t_1}^{t+t_2} a_{tj} \\ & P_t^\phi \geq a_{tj}, j \in t+t_1, \cdots, t+t_2 \end{aligned} \tag{5.15}$$

For the until operator, we define extra slack variables similar to [13] in order to make the constraints linear in terms of the variables. The constraints for the until operator could be interpreted as follows:

$$P_t^\phi = \overset{[}{j} = t+t_1]t+t_2 \bigvee (\wedge_{k=t}^{k=j-1} P_k^p) \wedge P_j^q)$$

Using this approach, we translate the given high level specification in STL ($\mathbf{X}_{t_0} \models \phi$) to a set of mixed integer linear constraints. At the end, we add the constraint $P_0^\phi = 1$, i.e. the overall specification ϕ is satisfied. Since Boolean variables are only introduced when halfspaces are defined, the computation cost of MILP is at most

exponential to the number of halfspaces times the discrete steps N.

5.5 Runtime Monitoring and Self-Correction

Let N be the horizon of the planning trajectory, and let $X_r(t)$ and $U_r(t)$ be the reference states and control inputs for $t \in [1, N]$ respectively. Note that $X_r(t)$ and $U_r(t)$ could be obtained offline by solving the MILP in Problem 1. During runtime, two threshold parameters θ_{space} and θ_{time} are defined to monitor the runtime execution. θ_{space} and θ_{time} are the space and time tolerances we want to maintain for the execution sequence. At time t', we denote the observed states as $X_o(t)$, where $t \in [1, t']$. The predicted states $X_p(t)$ of the robot is generated based on the observed states and the reference inputs until the end of the execution ($t = N$), i.e,

$$\begin{aligned} X_p(\tau+1) &= f(X_p(\tau), U_r(\tau)), \quad \tau = t', \cdots N-1 \\ X_p(\tau) &= X_o(\tau) \quad \text{for} \quad \tau = 1, \cdots, t' \end{aligned} \tag{5.16}$$

Let $\mathbf{X}_t^p$ denote the predicted trajectory at time t, we then evaluate the tolerance $r_{time}(\mathbf{X}_t^p)$ and $r_{space}(\mathbf{X}_t^p)$ for the predicted trajectory. If at time t we have $r_{time}(\mathbf{X}_t^p) \geq \theta_{time}$ and $r_{space}(\mathbf{X}_t^p) \geq \theta_{space}$, then it indicates the execution sequence is able to satisfy the specification and there is no need for correction. We simply use $U_r(t)$ from offline calculation as the control inputs at time t. Otherwise, the event-trigger MPC module will be activated and correct the execution.

5.5.1 Event-triggered Model Predictive Control

An event-triggered MPC is designed for runtime self-correction, where we are constantly evaluating whether the predicted trajectory still satisfies the given specification and maintains a specific tolerance degree. If $r_{space}(\mathbf{X}_t^p) < \theta_{space}$ or $r_{time}(\mathbf{X}_t^p) < \theta_{time}$ at time t, it suggests possible violations for the specifications in the future and the MPC module will be triggered. The MPC problem is formulated as follows:

$$\begin{aligned} \min_{\mathbf{X}(t),u(t)} \quad & \sum_{\tau=t}^{\tau=t+T} (X_r(\tau) - \mathbf{X}(\tau))^T Q (X_r(\tau) - \mathbf{X}(\tau)) \\ \text{subject to} \quad & \mathbf{X}(\tau+1) = f(\mathbf{X}(\tau), u(\tau)), \tau \in [t, t+T-1] \\ & \mathbf{X}(t+T) = X_r(t+T) \end{aligned} \tag{5.17}$$

where T is the horizon. By solving the MPC problem, we try to bring the robot back to the reference trajectory. Note that only the first step of the computed optimal control strategy (denoted as $u^*(t)$) is implemented, i.e, at time t, we use $u^*(t)$ instead of $U_r(t)$ as the control input. We will re-evaluate the predicted states at the next time step iteratively until the end of the planning trajectory.

5.6 Case Studies

In this section, we consider two different case studies, where the first one has tighter time constraints and the second one has tighter space constraints. The experiments are run through YALMIP-CPLEX on a computer with 2.8GHz processor

and 8GB memory. The MPC has a horizon T=10. For both examples, we use $\theta_{space} = 0.3$ and $\theta_{time} = 4$.

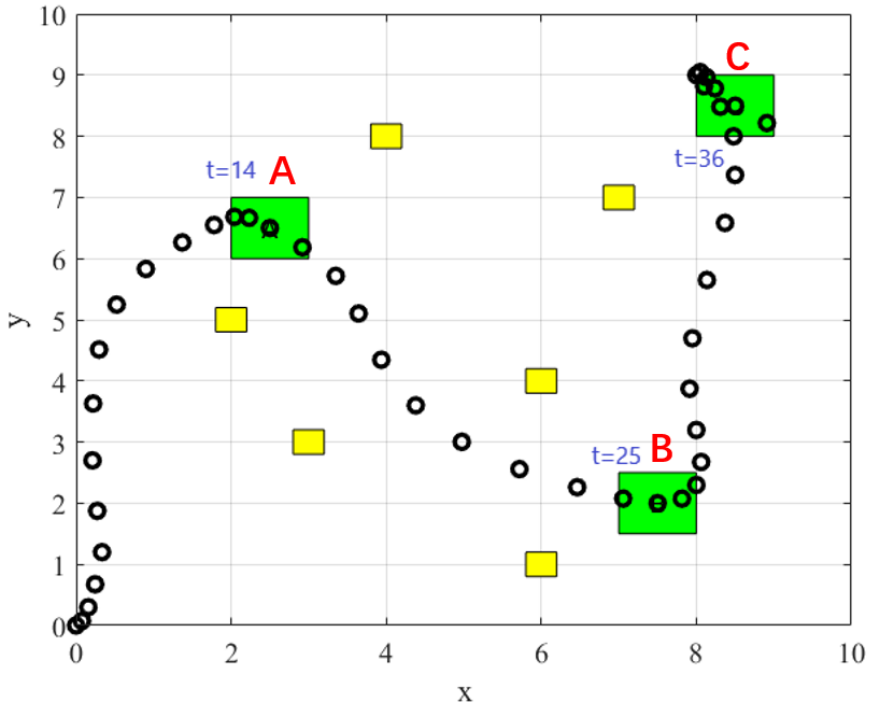


Figure 5.2: Resulting path with maximum space and time tolerances for ϕ_1. The blue text shows the time that the robot enters each green region.

We first consider a sequential task that the robot needs to visit position A between 10 and 20 seconds, and visit B between 21 and 31 seconds, visit C between 32 and 42 seconds, and never be in the yellow regions O_is ($i \in [1, k_1]$, where k_1 is the number of obstacles). The STL specification is given as below.

$$\phi_1 = \Diamond_{[10,20]} A \wedge \Diamond_{[21,31]} B \wedge \Diamond_{[32,42]} C \wedge (\underset{i=1,\cdots,k_1}{\wedge} \Box \neg O_i) \tag{5.18}$$

Region A is represented as $(x > 2 \wedge x < 3 \wedge y > 6 \wedge y < 7)$ and similarly for other regions. The optimization problem is formulated as in equation (5.3). We

assume the velocity information is perfect when we are generating the reference trajectory offline, and is not perfect with a white noise deviation added during runtime execution. The resulting reference trajectory is shown in Fig 5.2. As can be seen from Fig 5.2, the final path of the robot stays far away from the yellow regions, and always goes through the center of the green regions for maximum space robustness. The robot also slows down when it enters green regions to maximize time robustness. Fig 5.3 shows that when the disturbance is small, the robot is able to still satisfy the specification without any correction.

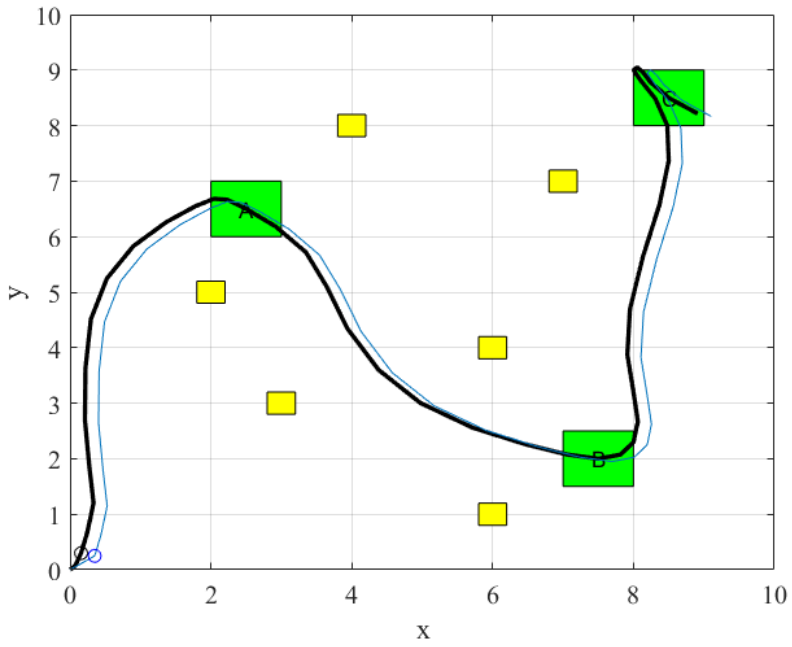


Figure 5.3: Monitoring runtime sequence (blue line) with space and time tolerances. The monitor indicates that the runtime sequence also satisfies ϕ. No correction is needed and MPC never turns on.

However, when the deviation is large, the MPC module will be turned on and guide the robot to satisfy the desired specification with self-corrections. Note that the blue dashed line in Fig 5.4 is the predicted trajectory at $t = 8$, and it is not able

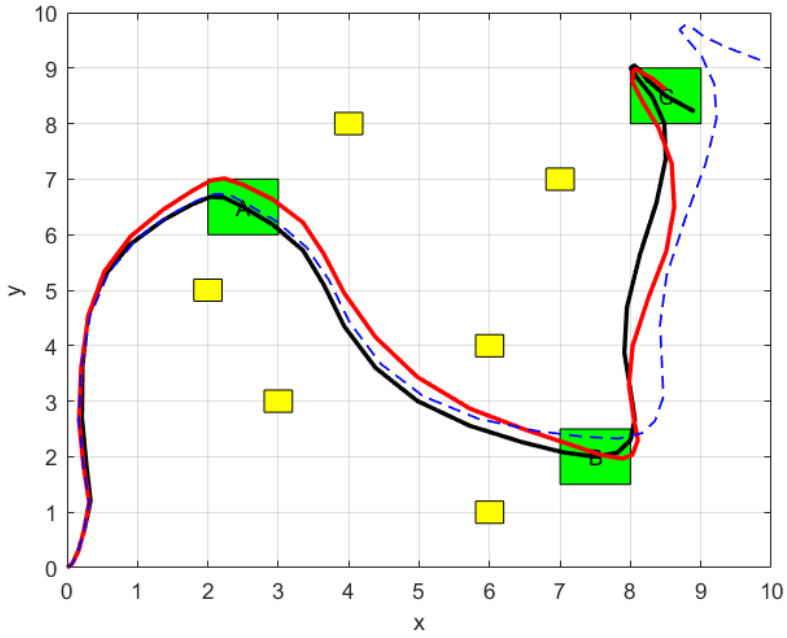


Figure 5.4: Resulting trajectory for ϕ_1 with self-correction. The blue dashed line indicates the predicted path at $t = 8$. The red line shows the path with self-corrections. The reference trajectory is marked in black.

to reach position C thus violating the specification. Fig 5.5 shows the triggering instances of MPC, and the MPC module has been triggered for 4 seconds in total in this example.

In the second example, we consider an environment with more obstacles but with a relatively looser time constraints. The specification is given as below, where we require the robot to eventually visit position A between 10 seconds and 20 seconds, and eventually visit position C between 32 seconds and 42 seconds while avoiding all k_2 obstacles.

$$\phi_2 = \Diamond_{[10,20]} A \wedge \Diamond_{[32,42]} C \wedge \left(\bigwedge_{i=1,\cdots,k_2} \Box \neg O_i \right) \tag{5.19}$$

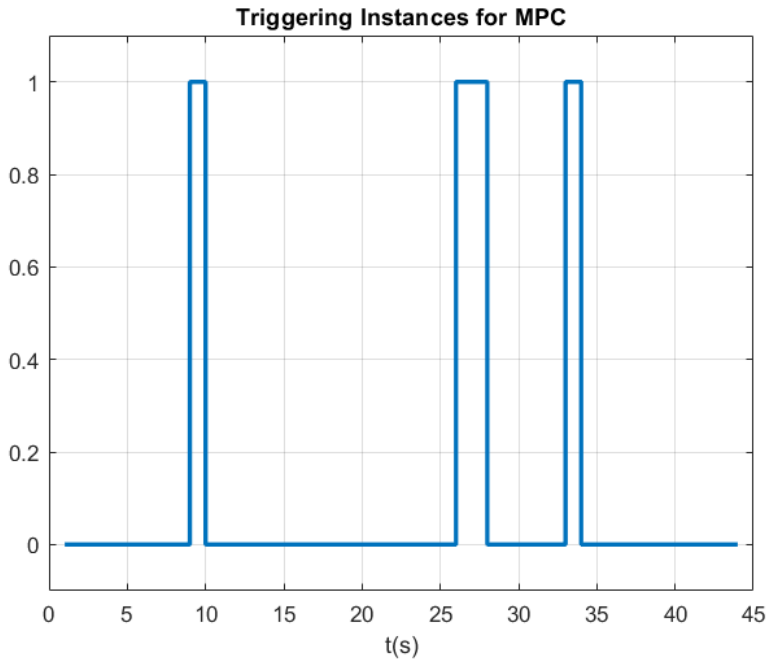


Figure 5.5: Triggering instances for MPC. The MPC module has turned on for 4 seconds in total.

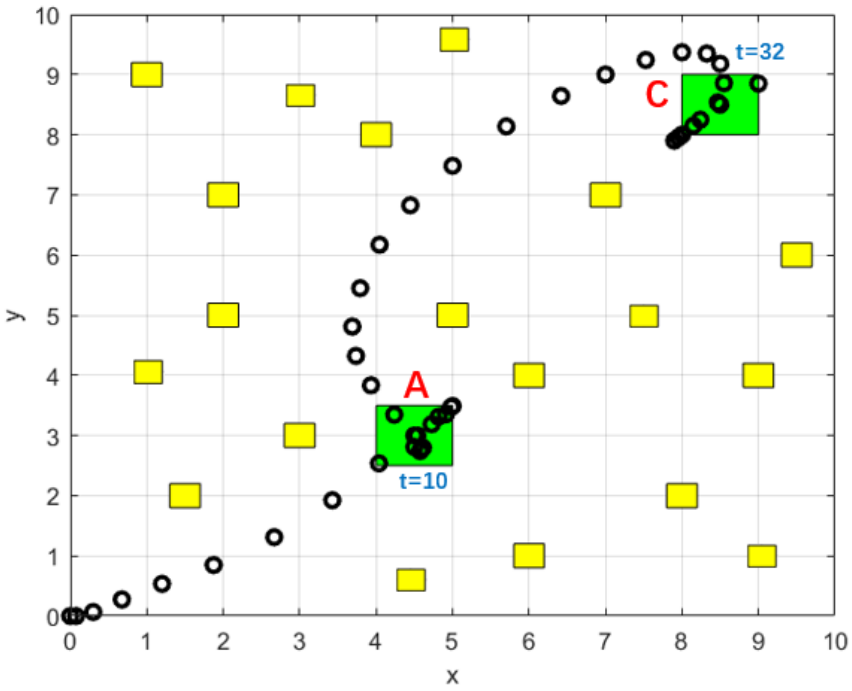


Figure 5.6: Resulting path with maximum space and time tolerances for ϕ_2

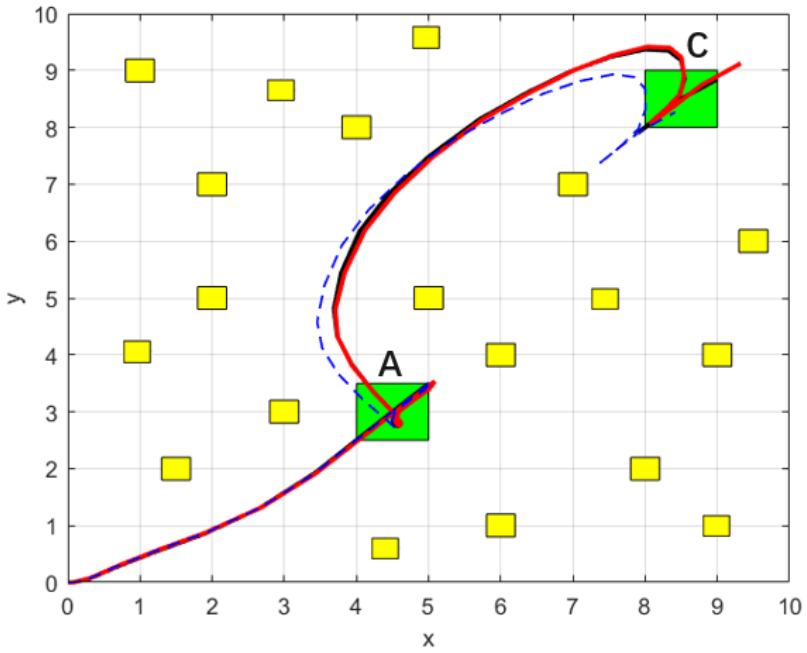


Figure 5.7: Resulting trajectory for ϕ_2 with self-correction. The blue dashed line indicates the predicted path at $t = 6$. The red line shows the path with self-corrections.

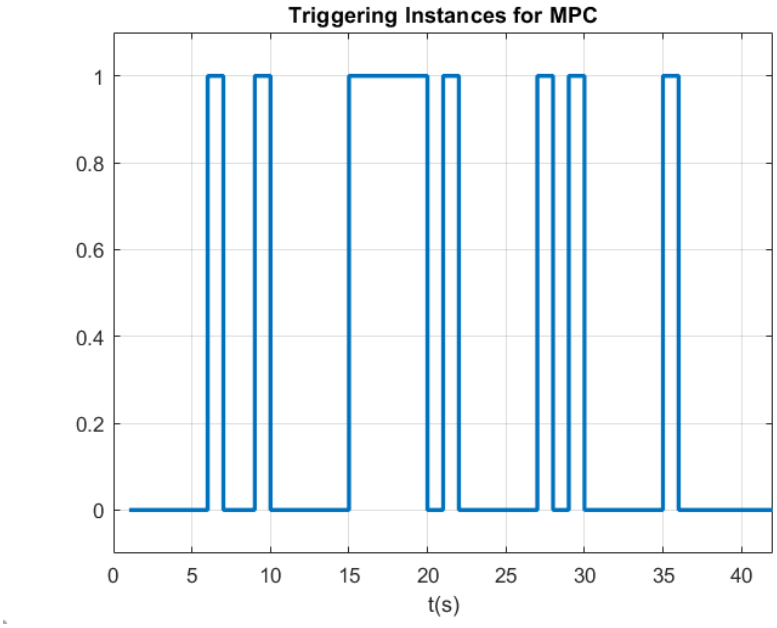


Figure 5.8: Triggering instances for MPC. The MPC module has turned on for 11 seconds in total.

Similarly, the offline planning is able to generate a path that maximize the space and time tolerances as shown in Fig 5.6. It is clearly visible in this case that the trajectory tends to stay in green regions as long as possible during the required time interval for maximum time robustness. During runtime execution, the blue dashed line in Fig 5.7 is the predicting trajectory at $t = 6$, and it reaches position C at the last time step. The time robustness requirement is thus violated and therefore MPC is triggered. Fig 5.8 shows the triggering instances of MPC, and MPC has been triggered for 11 seconds in total in this example. Compared to the first example, MPC has been triggered more frequently due to the complexity of the environment (It is more likely to hit an obstacle). Table 1 summarizes the number of linear constraints and computation time for each of the examples. Both examples have similar computational complexity since the first example has a more complicated task and the second example has a more complicated environment.

STL Specifications	# of linear constraints	Computation time (s)
ϕ_1	3154	10123
ϕ_2	2615	9673

Table 5.1: Number of constraints and computation time

5.7 Chapter Summary

In this chapter, we have presented an optimization-based approach for robot planning, monitoring and self-correction problems under STL specifications with finite time constraints. Our approach translates the STL specifications into mixed-

integer linear constraints, and the goal of the optimization problem is to maximize the overall space and time tolerances under double integrator dynamics of the robotic agent. During runtime execution, we consider a realistic situation where the velocity information is not perfect. A prediction module and a self-correction module with event-triggered model predictive control have been designed to predict and prevent possible future violations of the specifications. The simulation results show promising performance of our approach to find an optimal solution, and the robotic agent is able to make self-corrections during runtime execution when the velocity information is noisy.

Since we have used a binary variable (z) with each halfspace, the problem would be complex if the environment contains too many halfspaces. Therefore, the future directions of this work could include task decomposition and reduction of binary variables. Other aspects such as learning from the self-corrections, and multi-robot cooperative planning could also be possible extension of this work.

Chapter 6: Statistics-Based Slippage Prediction and Correction with Object Classification using a Dexterous Robotic Hand

Slip detection and correction plays a very important role in robotic manipulation tasks, and it has long been a challenging problem in the robotic community. In this chapter, we propose a complete framework to predict, detect and correct slippage through the use of BioTac SP sensors attached to a five fingered dexterous robotic hand. We take advantage of both the haptic data from these sensors, combined with synchronized video data from a camera, to apply a statistical based approach to slip prediction and correction, and the ability to classify objects as either rigid or soft in order to prevent over exertion of the robotic grasp. We tested our algorithm by adding weight to the empty container which was initially held stably by the robotic hand, and our algorithm is able to predict and prevent the slip for different types of containers.

6.1 Related Work

When grasping an object, humans are able to prevent the object from slipping from their grasp by constantly adjusting their grip [32]. This is possible due to our highly sensitive slip detection capabilities. However, the ability for a robotic hand

to grasp an object, detect when that object is slipping, and correct for that slippage without exerting too much force on the object is difficult still today especially while the weight of the object is changing at an unknown rate. In this work, the ability for a robotic hand to autonomously detect the moment of slippage and be able to stop this slippage without crushing the object was investigated using SynTouch BioTac SP sensors attached to a five fingered robotic hand, the Shadow Dexterous Hand. A statistics-based approach was used to detect the moment of slip, and a weight estimation technique was used in order to apply enough force to stop the object from falling while the weight of the cup was changing in real time. Lastly, the object in the hand was classified as either rigid or soft so that the robotic hand would adjust the correction algorithm based on the classified object. This was done in order to prevent the robotic hand from exerting forces that would crush an object while trying to prevent slippage.

The benefit of having a slip detection and correction through robotic hands is immense. In [33], the authors state that there is a crucial need for limb-absent people to have their artificial hands be able to detect and correct object slippage when they do not have direct feedback from the robotic hand sensors. Our work provides the ability for the robotic hand to autonomously detect object slippage and correct for that slippage while preventing deformation of the object. As discussed in [34], it is important for robotic hands to be able to pick up or catch an object with an unknown mass and friction where the goal is to reliably hold the object without the object slipping or over correcting for slippage and crushing the object. Therefore, there is a need and use for dexterous hands that implement a slippage

detection and correction algorithm to prevent slippage while having the ability to know the composition of the objects being held in the hand which was our goal of this work.

The contributions of this work are as follows. First, we've collected a data-set of synchronized haptic and vision data for slippage detection and correction problems with BioTac SP tactile sensors and a high resolution camera. Second, we analyzed the data-set to determine the empirical distribution and the correlation of the haptic data around the moment of object slip. We discovered a certain pattern in the correlation data analysis, and a median absolute deviation (MAD) method is used to predict the slip time. Finally, we propose a control framework for slippage detection and correction based on the sensor data in order to stop object slippage in the robotic hand. As part of our control framework, the robotic hand is able to correct object slippage and uses previous knowledge of the structure of the object in order to prevent crushing the object. Self-managed safety and correctness are essential for autonomy which is why it was crucial for our robotic hand to implement the control algorithm autonomously.

6.2 Related Work

Over the years, there has been research that focused on using haptic data to detect the moment an object starts slipping from a robotic end effector. The need of haptic sensors that mimic the sensory capabilities of the human hand has been well documented by N. Wettles, Jeremy A. Fishel and Gerald E. Loeb in [45] with their

discussion of their first BioTac sensor. There has been many experiments employing some version of the BioTac sensors in order to detect and correct for object slippage.

In [38], researchers use the Pac (vibrational pressure) readings from BioTac sensors attached to a three finger manipulator to detect micro-vibrations in order to detect slippage. If 11 out of the 22 pressure samples in a time window were above a certain threshold then a slippage was detected. For our experiment, the real time data from 24 electrodes in each of the five fingers in a time window were used to see if their data is correlated with the average electrodes data at the moment of slippage over all slip experiments. Also, this previous paper did not go into stopping a cup from slipping from the robotic hand while weight is being continuously added to the cup instead the weight was changed while the object was on a table and then lifted up.

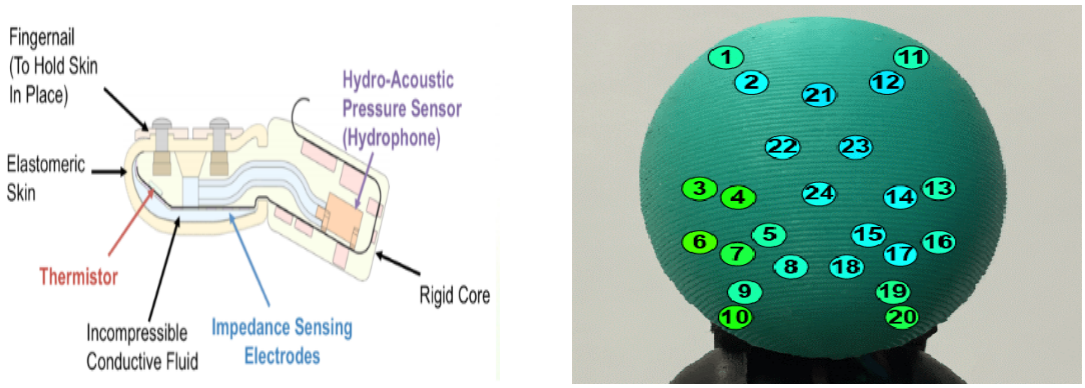
There have been many papers using neural networks to decipher the BioTac sensor readings in order to build their detection and correction algorithms. In [33], M. Abd et al. constructed their slippage detection algorithm using data from BioTac SP sensors and the application of an artificial neural network classifier. Their approach focused more on classifying the direction of an object slipping from a grasp. Our approach relies on the statistics of the sensor data around the slippage time which allows us to have a high accuracy of slippage detection while requiring a much smaller data-set. We also focused more on preventing an object from slipping. The approach of N. Wettels et al. [46] focused on using the BioTac electrode readings to estimate the tangential and normal forces applied by the Otto Bock M2 hand to a styrofoam cup. The cup was filled with water during their grasp control experiments

in order to see if the robotic fingers could correct for slippage caused by the continuous increased weight of the flowing water. Using their grasp control algorithm, it was possible to correct for the slippage of the soft cup, but as noted by the authors, their robotic hand had issues with correcting for slippage caused by a rapid fill rate. For our experiment, our robotic hand is able to correct for the slippage of rigid and soft cups regardless of the rate of fill.

In [47], the authors used a Long Short Term Memory (LSTM) network to detect slippage with tactile sensors. A cutoff significance threshold is set manually, and slippage is detected if the output of the LSTM network exceeded the threshold. The neural network approach is ad-hoc and relies on selecting a good cut-off threshold and window size. With our approach, we discover a statistical pattern using correlation analysis for the slip data which provides a more accurate detection of the moment of slip as compared with the LSTM method.

(a) Experiment Configuration (b) Funnel and Bb pellets

Figure 6.1: Experiment setup for collecting haptic-vision dataset. Bb pellets are used as the filler and we use a funnel to ensure a constant pouring rate



(a) BioTac Sensor Schematic (b) Timed Automaton for the checker

Figure 6.2: BioTac Sensor Schematic and Electrode Locations

6.3 Data Description

Eighty sets of synchronized haptic and visual data were collected for the slippage detection and correction experiment with different containers. Sixty four of the experiments were conducted with a rigid container and the rest with soft containers. The schematic of the BioTac SP, a haptic sensor, is shown in figure 6.2. A high resolution camera was used to record the visual data.

For each experiment, a bottle is initially securely held by the BioTac SP sensors that are attached to the Shadow Hand fingers. The bottle is held at the same height at the beginning of each experiment. During the experiment, weight is added to the bottle at a consistent rate by pouring bb pellets into the bottle. The experiment is run until the bottle slips completely from the hand. The visual data is synchronized with the haptic data using the ROS platform [35] during the data collection process. We additionally labelled the dataset with the frame number where the slippage occurs.

6.3.1 Haptic Data

Haptic data is sampled at 100HZ. Each sample has a dimension of 5×29, where 5 is the number of fingers and 29 is the number of sensor data for each finger. There are 24 electrodes on each finger which are located as shown in figure **??**. The types of the sensor data are summarized in figure 6.3.

Sensory Modality	Symbol	Range	Resolution	Frequency Response
Impedance	E_n	0 - 3.3V	3.2 mV	0 - 100 Hz
Fluid Pressure	P_{DC}	0 - 100 kPa	36.5 Pa	0 - 1040 Hz
Microvibration	P_{AC}	+/-0.76 kPa	0.37 Pa	10 - 1040 Hz
Temperature	T_{DC}	0 - 75 C	0.1 C	0 – 22.6 Hz
Thermal Flux	T_{AC}	0 - 1 C/s	0.001 C/s	0.45 – 22.6 Hz

Figure 6.3: BioTac Sensor Data Types

6.3.2 Visual Data

A high resolution RGB camera (Logitech C922x) is used for collecting the visual data. The camera is positioned at the side of the shadow hand, such that the cup and robotic hand are in the middle of the image. Visual data is sampled at 30 frames per second. The resolution of the image is 1920×1080. The experimental configuration is shown in 6.1.

6.3.3 Indexing of the data

Since the sampling rate of haptic data and vision data is different, at time t seconds of the experiment, the index of haptic data is $100 \cdot t$, and the index of vision

data is $30 \cdot t$. Given a frame number I_v, the corresponding haptic data index I_h is given by $I_h = [\frac{100 \cdot I_v}{30}]$, where $[\cdot]$ is the nearest integer operator.

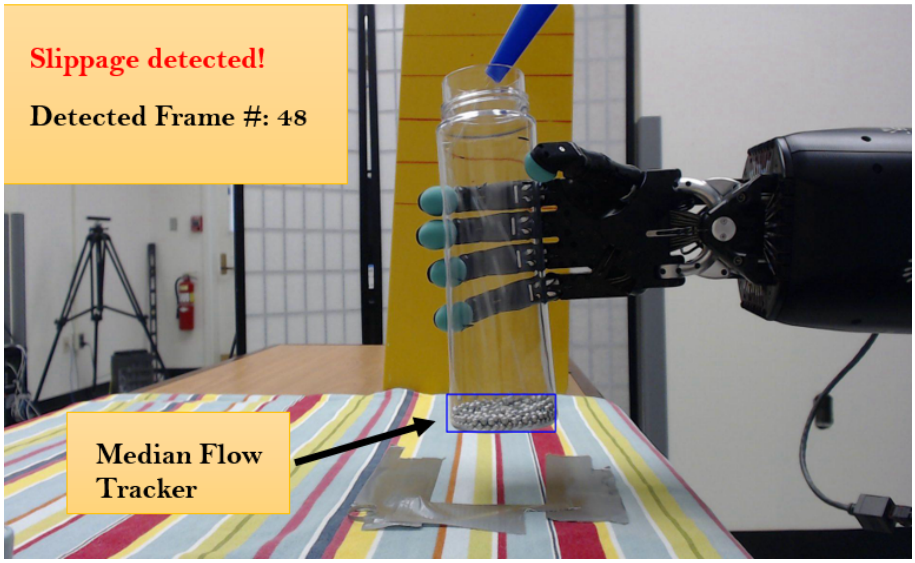
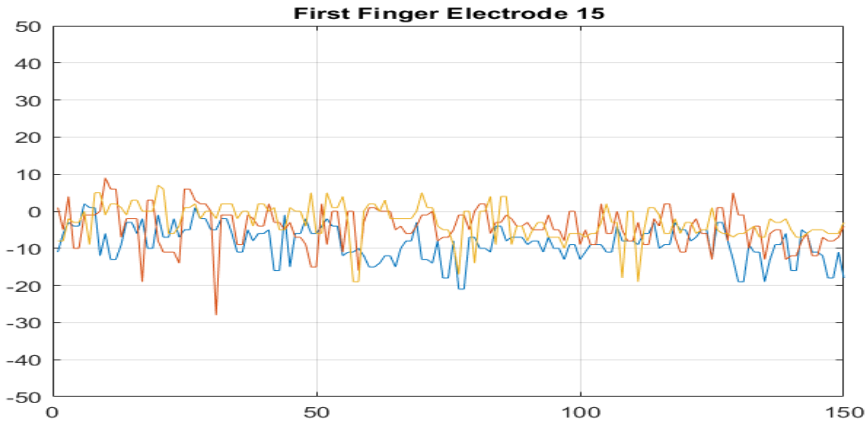


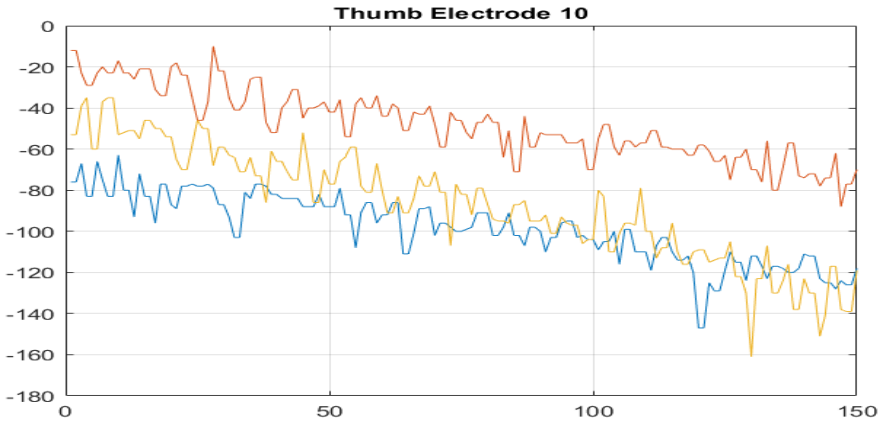
Figure 6.4: Median Flow Tracker is used to determine t_j^*. In this experiment, first slippage is detected at frame 48 and t_j^*=1.6s

6.3.4 Data Pre-Processing

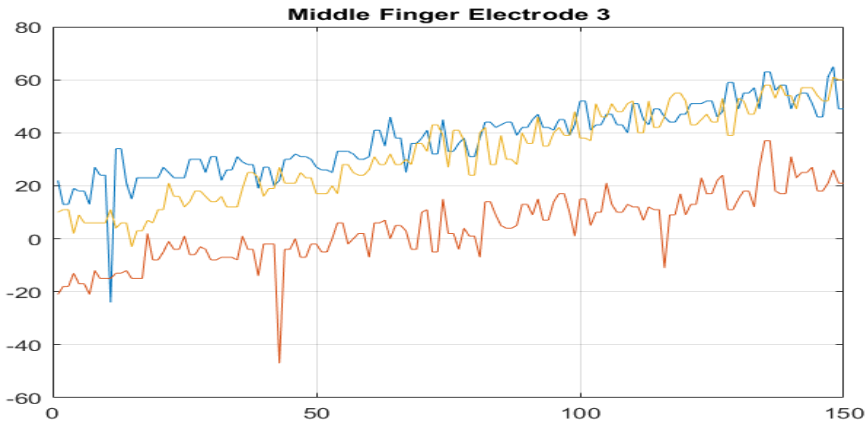
In order to correct or prevent slippage, it is important to understand the statistics of the sensor data before, during and after the moment of slip. We first use a median flow tracking algorithm [36] to determine the time that the cup starts to slip for each experiment of the dataset. The median flow tracker tracks the object in both forward and backward directions in time and measures the discrepancies between these two trajectories. Minimizing this Forward-Backward error enables the tracker to reliably track the object. We use the tracker to track the object grasped in the hand and detect if slippage occurs based on the velocity of the object. As shown in figure 6.4, the blue bounding box indicates that we are tracking



(a) electrode 15 on first finger (FF)



(b) electrode 10 on thumb (TH)



(c) electrode 3 on middle finger (MF)

Figure 6.5: 150 electrode data samples on FF, TH and MF, before t^* for three different experiments. The sensor data has been normalized by subtracting the values at rest.

the bottom of the cup. The text on the top left corner indicates the status of the experiment, whether slippage is detected or not, and the first frame that slippage is detected. We claim that slippage occurs if the velocity of the object is greater than some small threshold ϵ. We denote t_j^* as the first moment that slippage occurs for the jth experiment.

6.3.5 Tactile Data Aggregation

Empirical Distribution Around Slippage Time t_j^*: Once we have determined the slippage time t_j^* for each $j \in \{1 \cdots 64\}$, we then align our data based on t_j^* since we're interested in the statistics around the slippage time. We denote the electrodes data as $x_{i,k}^j(t)$, where j is the index for the experiment number, $i \in \{1 \cdots 5\}$ is the finger index, $k \in \{1 \cdots 24\}$ is the electrode index, and t is the time index. The empirical distributions of the electrodes data for all 64 experiments $x_{i,k}(t)$ are calculated for $t \in [t_j^* - t_a, t_j^* + t_b]$, where t_a and t_b are the constants to represent the time around slippage occurs. '

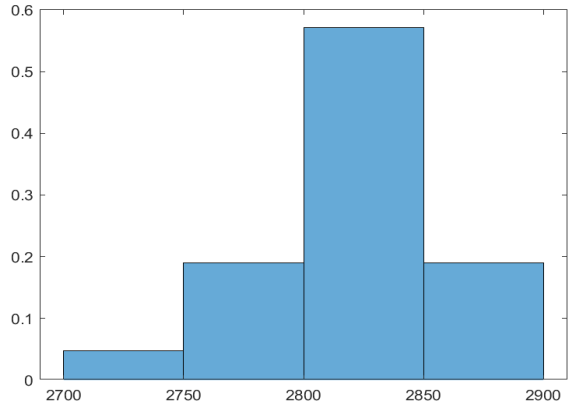
It is not necessary to use all 24 dimensions of tactile data, since some of the sensors may not be in contact with the object during the slippage time interval. We evaluate the empirical distribution of each electrode data in the time interval $[t_j^* - t_a, t_j^* + t_b]$, and we select the top five electrodes with most impact. We evaluate the impact using entropy. The top 5 electrodes with maximum entropy empirical distributions will be selected for the cross-correlation computation.

$$H(X) = -\sum_{i} P_i log(P_i) \tag{6.1}$$

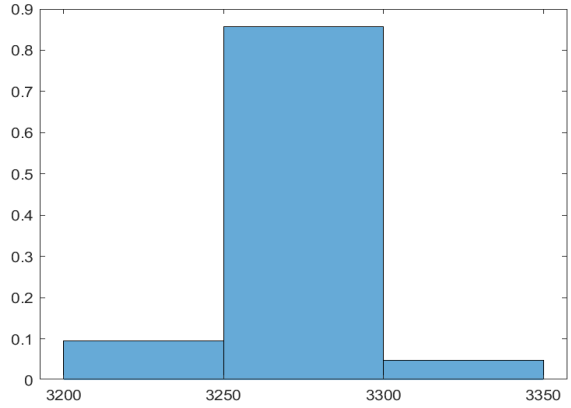
6.4 Slippage Detection Algorithm

Once we have determined the slippage time t_j^* for each experiment $j \in \{1 \cdots M\}$, where M is the total number of experiments in the dataset, we then align our data based on t_j^* since we are interested in the statistics around the moment of slip. We denote the electrodes data as $x_{i,k}^j(t)$, where j is the index for the experiment number, $i \in \{1 \cdots n_f\}$, is the index of the finger , $k \in \{1 \cdots n_e\}$ is the electrode index, and t is the time sample index. n_f is the number of fingers which is equal to 5 for the Shadow robotic hand, n_e is the number of electrodes which is equal to 24 for the BioTac SP sensors attached to the Shadow Hand. Finger index 1 to 5 corresponds to first finger (FF), middle finger (MF), ring finger (RF), little finger (LF) and thumb (TH), respectively.

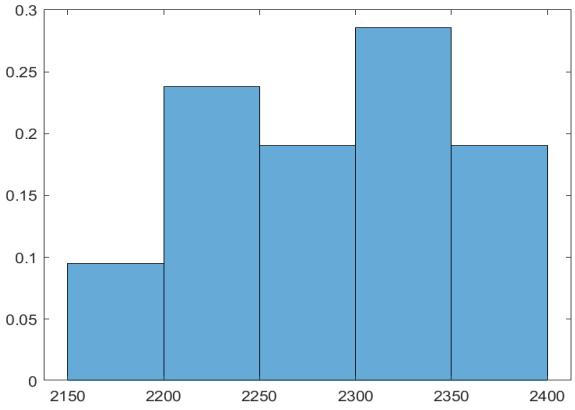
Figure 6.5 demonstrates the electrode data on different fingers for three different experiments. Figure 6.5 shows the electrodes data for time interval $I = [t_j^* - t_a, t_j^* + t_b]$, where t_a and t_b are the integer constants to represent the time interval around when slippage occurs. For different experiments, some electrodes on the fingers behave similarly before slippage occurs. For example as shown in figure 6.5(b), the readings for electrode 10 on the thumb decreases when approaching t^* for all three experiments. However, some electrodes will behave differently for each experiment. Some electrodes' value will not drastically vary while some will



(a) Distribution for $x_{1,1}(t_1^*)$



(b) Distribution for $x_{1,7}(t_1^*)$



(c) Distribution for $x_{1,16}(t_1^*)$

Figure 6.6: Empirical Distributions for three different electrodes on first finger at t_j^*

increase and some will decrease, as shown in figure 6.5 (a), (b) and (c). Therefore, we propose a statistical based method to detect slippage. This method calculates the correlation between the realtime haptic data sequence X_R and the pre-slip haptic data sequence X_H. The dimension of X_H is $X_H \in I \times n_f \times n_e$, where I is the window size $I = (t_b + t_a + 1)$. A single pre-slip haptic data sequence X_S for the jth experiment can be defined as follows:

$$X_S(t, j, i, k) = x^j_{i,k}(t) \tag{6.2}$$

for $t \in [t^*_j - t_a, t^*_j + t_b]$, $i \in [1, n_f]$, $k \in [1, n_e]$. Then, we can calculate the average pre-slip haptic data sequence of the dataset as follows:

$$X_H(t, i, k) = \frac{1}{N_e} \sum_{j=1}^{N_e} X_S(t, j, i, k) = \frac{1}{N_e} \sum_{j=1}^{N_e} x^j_{i,k}(t) \tag{6.3}$$

for $t \in [t^*_j - t_a, t^*_j + t_b]$, $i \in [1, n_f]$, $k \in [1, n_e]$, $N_e = 64$ is the total number of experiments in the dataset.

The realtime haptic data sequence X_R has the same dimension as X_H, and it is updated with every new sample. Let t_{now} denote the current time index, then

$$X_R(t, i, k) = x^j_{i,k}(t) \tag{6.4}$$

for $t \in [t_{now} - (t_a + t_b), t_{now}]$, $i \in [1, n_f]$, $k \in [1, n_e]$.

6.4.1 Slippage Detection using Correlation Coefficient of Two Time Series

The correlation between the realtime sequence X_R and the pre-slip sequence X_H for electrode k on finger i at time t can be calculated as follows.

$$\rho_{i,k}(X_R, X_H) = \frac{1}{N-1}\sum_{j=1}^{N}(\frac{X_{R_j} - \mu_{X_R}}{\sigma_{X_R}})(\frac{X_{H_j} - \mu_{X_H}}{\sigma_{X_H}}) \tag{6.5}$$

where μ and σ are the mean and standard deviation of the haptic data sequence for electrode k on finger i. $N = (t_a + t_b + 1)$ is the window size or number of observations in the sequence. We select $t_a = t_b = 75$ (which is equivalently 0.75 seconds) in order to understand the statistics around the moment of slippage. $(N-1)$ here is the Bessel's correction, which uses $(N-1)$ instead of N in the formula for the sample variance and sample standard deviation. This method corrects the bias in the estimation of the population variance. The total correlation is a sum of all the electrode correlations.

$$\rho(X_R, X_H) = \sum_{i=1}^{5}\sum_{k=1}^{24}\rho_{i,k}(X_R, X_H) \tag{6.6}$$

During the testing phase, we calculate the cross-correlation between the real-time haptic data and the haptic data that we collected for the interval around the moment of slippage. From our experiments, we notice that at the moment of slippage t^*, there is always a peak outlier appearing in the cross-correlation sequence,

as shown in Figure 6.7. Therefore, we could transfer our slippage prediction problem into a peak outlier detection problem. A peak in the cross-correlation will suggest a possible slip, and the forces applied on the object by the fingers should be increased to prevent slippage.

This peak can be detected by measuring the median absolute deviation (MAD). In statistics, the median absolute deviation (MAD) is a robust measure of the variability of a univariate sample of quantitative data [44]. Let $C_1, C_2, \cdots, C_N$ be the correlation sums sequence for a window size of N. Let $\bar{C}$ denote the median of the sequence. Then the Median absolute deviation could be calculated as follows:

$$MAD = median(|C_i - \bar{C}|) \tag{6.7}$$

The MAD may be used similarly to how one would use the deviation for the average. In order to use the MAD as a consistent estimator for the estimation of the standard deviation σ , one takes

$$\tilde{\sigma} = k \cdot MAD \tag{6.8}$$

where k is a constant scale factor, which depends on the distribution. In this work we choose $k = 1/\Phi^{-1}(\frac{3}{4}) \approx 1.4826$ by assuming the correlation sequence is normally distributed during the experiment. For our experiment, we claim a point is detected as a peak outlier if $C_i > 3\tilde{\sigma}$. We selected different window sizes of 20, 50, 100, and

we compared the performance with different methods. The error is calculated as:

$$e = \frac{|t_{detect} - t^*|}{t_{total}} \tag{6.9}$$

where t_{detect} is the time index for the detected peak outlier, t^* is the ground truth from the dataset, and t_{total} is the total length of the experiment. To evaluate the performance of our algorithm, we compare the proposed algorithm with two other algorithms. We summarized the algorithms as follows:

- Corr-MAD-All (proposed algorithm): Compute the total correlation between the realtime sequence X_R and X_H.
- Corr-MAD-Single: Compute the total correlation between the realtime sequence X_R and single sequence X_S.
- DNN-LSTM: Detect the slippage using LSTM deep neural network.

For the DNN-LSTM approach, we construct a LSTM network similar to [47], and use X_R as input. We set a significance cut-off as 0.95. The results are shown in table 6.1. As can be seen from the results, the proposed algorithm shows the best performance. We also notice that for all different window sizes, Corr-MAD-ALL always performs better than Corr-MAD-single. This is due to the fact that with a randomly selected sequence, there are more uncertainties.

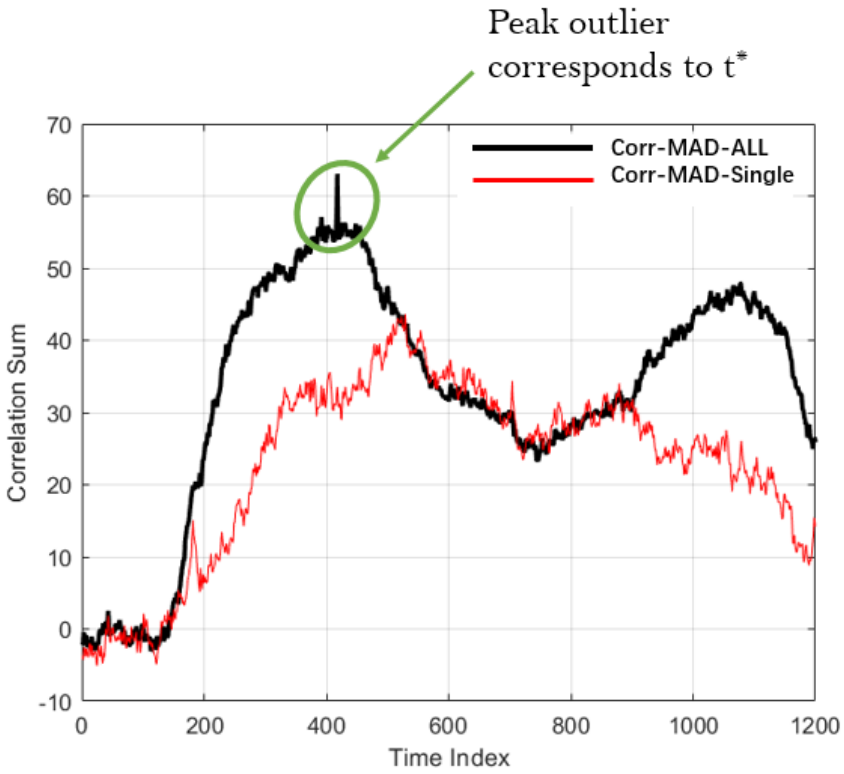


Figure 6.7: Comparison of the correlation sum when using average sample sequence from the dataset (black line) versus using a random single sample sequence from the dataset (red line).

6.5 Slippage Correction Algorithm with Weight Estimate

During the testing phase, we calculate the cross-correlation between the real-time haptic data and the haptic data we collected right before slippage occurs. A high correlation will suggest a possible slippage, and the forces applied on the object by the fingers should be increased to prevent slippage. For slippage correction, it is important that the robotic hand is not over correcting or crushing the object. To serve this purpose, we propose a weight-estimation algorithm and a PD controller for slippage correction.

Method Used	Window Size	Average Error
Corr-MAD-single	20	7.41
Corr-MAD-ALL	20	6.67
Corr-MAD-single	50	8.23
Corr-MAD-ALL	50	5.47
Corr-MAD-single	100	7.75
Corr-MAD-ALL	100	**2.77**
DNN-LSTM	100	15.57

Table 6.1: Comparison slippage prediction using different methods and window size

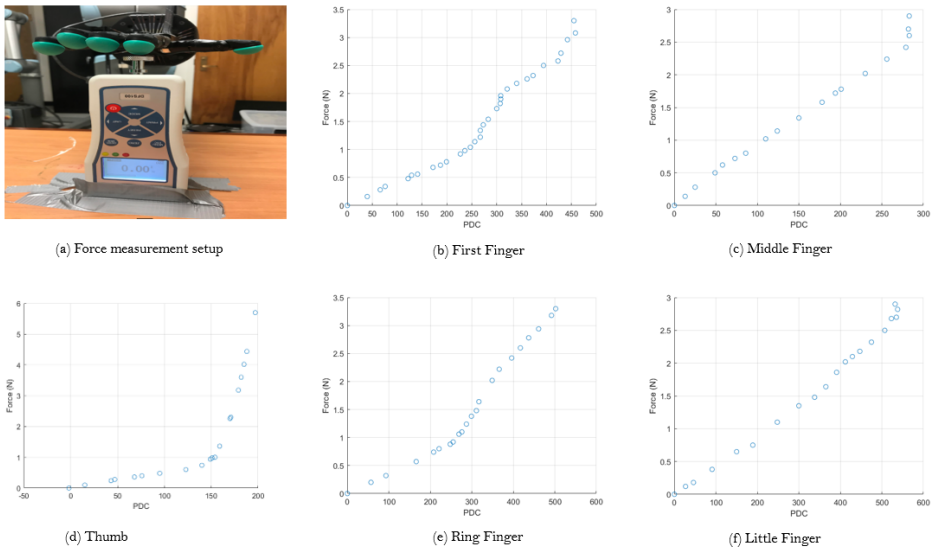
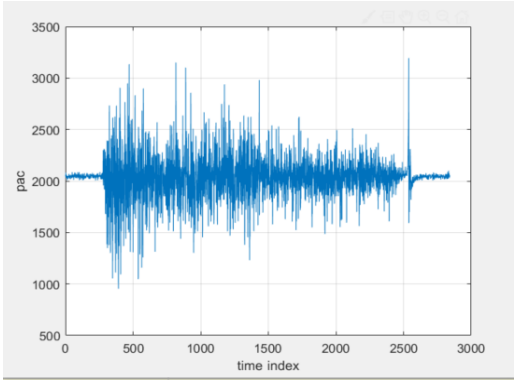


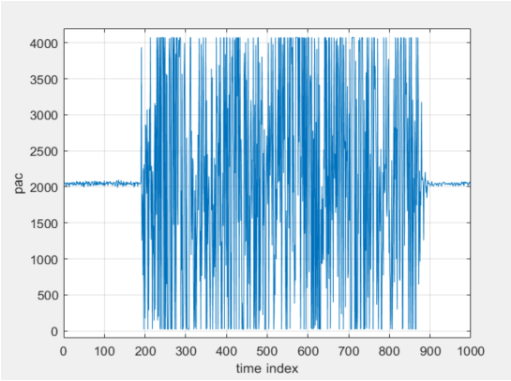
Figure 6.8: Force estimation from Biotac SP sensor pdc readings. (a) shows the force measuring setup. (b)-(f) show the force to pdc ratio for each of the finger. The pdc value is saturated at around 3N for all fingers. Below the 3N range, the pdc-force ratio is almost linear for all fingers except the thumb.

6.5.1 Weight Estimation

In order to augment our correction algorithm, it is important to estimate the weight of the grasped object and apply forces through the fingers accordingly. To estimate the weight of the grasped object, we leverage the vibration sensor, Pac,



(a) Pac for a rigid container



(b) Pac for a soft container

Figure 6.9: Vibration sensor readings (Pac) for different types of containers during one experiment

from the fingers. The vibration sensor readings are significant only when the bb pellets start to enter the container. As shown in Figure 1(b), the narrow funnel ensures that the pouring speed is almost always constant. Therefore, the weight increase from time t_1 to t_2 could be estimated as:

$$\Delta W_{t_1,t_2} = I_{P_{ac}(t)} \cdot \sum_{t=t_1}^{t=t_2} R \tag{6.10}$$

where I is an indicator function and takes a value 1 when the vibration sensor has a significant reading and 0 otherwise. R is a constant which decides the rate of the weight increase and depends on the size of the funnel.

6.5.2 Container Classification

Additionally, we classify the container into two different classes, i.e. rigid containers or soft containers. It is important to classify the container since we do not want to apply a grasping force that will destroy the container if the container

is soft or fragile. As shown in Figure 6.9, we noticed that the vibration sensor readings behave regularly for soft containers as compared to rigid containers. Once the container is detected as a soft container, an additional threshold for the controller will be set so that the container will not be crushed. This means that the robotic hand will correct for slippage up to the threshold. There will be a warning message displayed once this threshold is met, and the correction algorithm will be stopped. This threshold can be adjusted based on user preferences such as deforming the cup in order to stop the cup from slipping from the hand.

6.5.3 Mathematical Model for BioTac Sensor - Force Estimation

The BioTac sensor provides two different ways to estimate forces. One way is through the pdc values which gives direct feedback of the fluid pressure inside the silicone skin. The other way is from the electrodes which provide feedback of the applied forces based on the area in contact with the BioTac surface. The pdc readings from the BioTac SP sensors provide a estimate of the force applied to the BioTac sensor when in contact with an object. However, it is necessary to calibrate this pdc data to force data. Therefore, we collected the force data by pressing the BioTac sensors against a Nextech DFS100 force gauge as shown by the setup in 6.8(a). This way we were able to collect pdc versus force data for all five fingers. As shown in Figure 6.8(b)-(f), for all fingers the pdc value is saturated at around 3N. For the range between 0N and 3N, a linear correlation between pdc value and force is visible for all fingers.

(a) rigid container (b) soft container

Figure 6.10: Implementation of the slippage correction algorithm with a dexterous robotic hand. We tested the algorithm with different containers and in both cases the robot is able to prevent slippage.

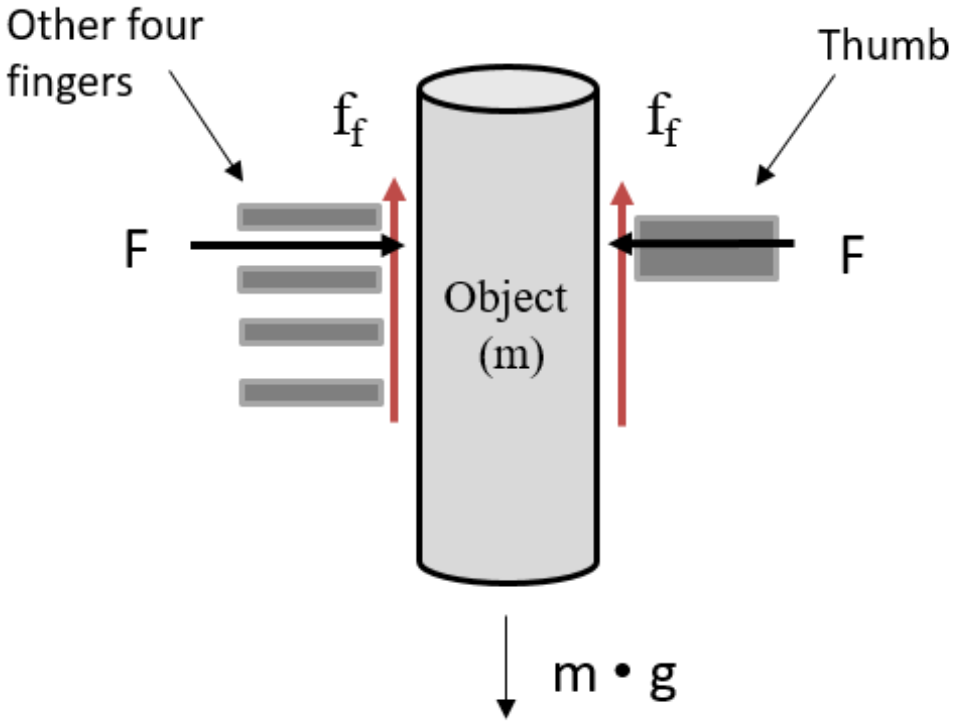


Figure 6.11: Simple Coulomb Friction Model

6.5.4 PD Controller

In this section, we use a simple Coulomb friction model as shown in Figure 6.11, where we assume the tangential force is proportional to the normal force. We assume the force applied by the thumb is equal to the combined force applied from the FF, MF, RF and LF.

$$f_f = \mu F \tag{6.11}$$

where μ is the friction coefficient which is different for different objects. F is the force applied by the thumb. Base on the results from previous section, the tangential force is then proportional to the Pdc values from the sensor.

We implemented a PD controller based on the fluid pressure values from the fingers.

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} \tag{6.12}$$

where $u(t)$ is the control signal, and it corresponds to the motor voltage for the finger joint. $e(t)$ is the error signal, which represents the difference between the desired force and the actual force applied on the object.

6.6 Chapter Summary

In this chapter, we investigated the slippage prediction, detection and correction problem using both haptic and visual data. We first described the data collection process for our experiments. A median flow tracking algorithm was used to determine the t_j^* for each experiment, and we analyzed the statistics for the haptic

data around the moment of slippage. We proposed a slippage prediction algorithm based on the median absolute deviation (MAD) of the correlation sequence, and a correction algorithm which automatically increases the grasping force based on weight estimation when the slippage has been predicted. Our algorithm shows high accuracy to predict and prevent the slippage. Furthermore, we leverage the vibration sensors to classify objects as either rigid or soft in order to prevent over exertion of the robotic grasp and prevent crushing the object.

Chapter 7: Conclusion

In this dissertation, we have addressed five problems on planning, self-monitoring and learning for robotic agents.

For the first problem, we propose a timed automata based approach for manipulator planning, using metric interval temporal logic (MITL). We have considered the automated generation of behaviors for a robotic manipulator while considering time constraints for both position changes (move from position A to position B) and for performing actions (grasping an object,releasing an object). We showed the execution sequence generated by UPPAAL for two different cases, and both of them satisfied the given specification.

Due to the uncertainty in the environment, the verification results obtained with respect to the system and environment models at design-time might not be transferable to the system behavior at run time. Therefore, for the second problem, we have considered the design of runtime monitors. We first modeled the manipulator as a hybrid system, thenwe dicussed the design of two runtime monitors. A model monitor is designed tomonitor the correctness of the task execution, and a safety monitor is designedbased on LTL3 specifications to guarantee the safety during the execution.

For the third problem, we present a modular Q-learning framework to deal with the robot task planning, runtime monitoring and self-correction problem. Given a specification with time con-straints, we construct the LTL3 monitor automaton, and a set of sub-task MITL monitors is then generated by decomposing and augmenting the monitor automaton. During the learning phase, a modular Q-learning approach is proposed such that each module could learn different sub-tasks. Our results show that during runtime, the sub-task MITL monitors could monitor the execution and guide the agent for possible self-correction if an error occurs.

Next, for the forth problem, we have presented an optimization-based approach for robot planning, monitoring and self-correction under STL specifications with finite time constraints. Our approach translates the STL specifications into mixed-integer linear constraints, and the goal of the optimization problem is to maximize the overall space and time tolerances under double integrator dynamics of the robotic agent. During runtime execution, we consider a realistic situation where the velocity information is not perfect. A prediction module and a self-correction modulewith event-triggered model predictive control have been designed to predict and prevent possible future violations of the specifications. The simulation results show promising performance of our approach to find an optimal solution, and the robotic agent is able to make self-corrections during runtime execution when the velocity information is noisy

Finally, we investigated the slippage prediction, detection and correction problem using both haptic and visual data. We first described the data collection process for our experiments. A median flow tracking algorithm was used to determine the

t^* for each experiment, and we analyzed the statistics for the haptic data around the moment of slippage. We proposed a slippage prediction algorithm based on the median absolute deviation (MAD) of the correlation sequence, and a correction algorithm which automatically increases the grasping force based on weight estimation when the slippage has been predicted. Our algorithm shows high accuracy to predict and prevent the slippage. Furthermore, we leverage the vibration sensors to classify objects as either rigid or soft in order to prevent over exertion of the robotic grasp and prevent crushing the object.

Bibliography

[1] Z. Lin and J. S. Baras, "Planning and Runtime Monitoring of Robotic Manipulator using Metric Interval Temporal Logic," IEEE International Systems Conference, 2019.

[2] Z. Lin, C. A. Meehan and J. S. Baras, "Statistics-Based Slippage Correction with a Dexterous Robotic Hand," Do Good Robotics Symposium, 2019.

[3] R. Alur, T. Feder, and T. A. Henzinger, "The benefits of relaxing punctuality," Journal of the ACM (JACM), vol. 43, no. 1, pp. 116– 146, 1996.

[4] T.A. Henzinger, "The theory of hybrid automata," Verification of Digital and Hybrid Systems, Springer, Berlin, Heidelberg, 2000.

[5] O. Maler, D. Nickovic, and A. Pnueli, "From MITL to timed automata," in Formal Modeling and Analysis of Timed Systems, ser. Lecture Notes in Computer Science, E. Asarin and P. Bouyer, Eds. Springer Berlin Heidelberg, 2006, no. 4202, pp. 274–289.

[6] Y. Zhou, D. Maity, and J. S. Baras, "Timed Automata Approach for Motion Planning Using Metric Interval Temporal Logic," European Control Conference (ECC 2016), 2016.

[7] A. Nikou, J. Tumova, and D.V. Dimarogonas, "Cooperative task planning of multi-agent systems under timed temporal specifications," In American Control Conference (ACC), 2016 (pp. 7104-7109).

[8] J. Tumova, and D.V. Dimarogonas, "Decomposition of multi-agent planning under distributed motion and task LTL specifications," In Decision and Control (CDC), 2015 IEEE 54th Annual Conference on, pp. 7448-7453. IEEE, 2015.

[9] J. Tumova and D.V. Dimarogonas, "A receding horizon approach to multi-agent planning from local LTL specifications," In Proceedings of the American Control Conference, pages 1775–1780, 2014.

[10] I. Filippidis, D.V. Dimarogonas, and K.J. Kyriakopoulos, "Decentralized multi-agent control from local LTL specifications," In Proceedings of the IEEE Conference on Decision and Control (CDC), pages 6235–6240, 2012.

[11] A. Aniculaesei, D. Arnsberger, F. Howar, and A. Rausch, "Towards the Verification of Safety-critical Autonomous Systems in Dynamic Environments," Theor. Comput. Sci., 232:79–90, 2016.

[12] S. L. Smith, J. Tumova, C. Belta, and D. Rus, "Optimal path planning under temporal logic constraints," in 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2010, pp. 3288–3293.

[13] S. Karaman and E. Frazzoli, "Vehicle routing problem with metric temporal logic specifications," 47th IEEE Conference on Decision and Control, CDC 2008.

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

[15] M. Kloetzer, C. Belta, "A fully automated framework for control of linear systems from temporal logic specifications," IEEE Transactions on Automatic Control, vol. 53, no. 1, pp. 287-297, 2008.

[16] E. Wolff, U. Topcu, and R. M. Murray. Optimization-based trajectory generation with linear temporal logic specifications. In International Conference on Robotics and Automation (ICRA), pages 5319–5325. IEEE, 2014.

[17] K. Kant and S. W. Zucker, "Toward efficient trajectory planning: The path-velocity decomposition," The International Journal of Robotics Research, vol. 5, no. 3, pp. 72–89, 1986.

[18] A. P. Sistla and E. M. Clarke, "The complexity of propositional linear temporal logics," Journal of the ACM (JACM), vol. 32, no. 3, pp. 733–749, 1985.

[19] S. Mitsch, K. Ghorbal and A. Platzer, "On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles," Proceedings of Robotics: Science and Systems, Berlin, Germany, 2013.

[20] S. Mitsch A. Platzer, "ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models," Runtime Verification - 5th International Conference, Toronto, ON, Canada, September, 2014.

[21] J. Tsitsiklis, Asynchronous stochastic approximation and Q-learning, Machine Learning APA, 1994.

[22] R.S. Sutton, A.G. Barto, Reinforcement learning: An introduction Volume 1, MIT press Cambridge, 1998.

[23] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner independent interface layer," in Int. Conf. on Rob. and Automation. IEEE, 2014.

[24] A. Pnueli, "The temporal logic of programs," in Proc. 1977 18th Annu. Symp. Found. Comput. Sci., 1977, pp. 46–57.

[25] J. Ouaknine and J. Worrell, "Some recent results in metric temporal logic," in Formal Modeling and Analysis of Timed Systems. Springer, 2008, pp. 1–13.

[26] Y. Zhou, D. Maity, and J. S. Baras, "Optimal Mission Planner with Timed Temporal Logic Constraints", European Control Conference (ECC 2015), 2015.

[27] K. He, M. Lahijanian, L.E. Kavraki and M.Y. Vardi, "Towards Manipulation Planning with Temporal Logic Specifications", IEEE Intl. Conf. Robotics and Automation (ICRA), 2015.

[28] S. Karaman, R. G. Sanfelice, and E. Frazzoli, "Optimal control of mixed logical dynamical systems with linear temporal logic specifications," in 47th IEEE Conference on Decision and Control. IEEE, 2008, pp. 2117–2122.

[29] Robotics e-Manual: OpenManipulator, `http://emanual.robotis.com/\docs/en/platform/openmanipulator/`

[30] H. Ding, M. Zhou, and O. Stursberg, Optimal Path Planning in the Workspace for Articulated Robots using Mixed Integer Programming, IEEE/RSJ International Conference on Intelligent Robots and Systems, October 11-15, 2009 St. Louis, USA.

[31] S. Saha and A. Julius, "Task and Motion Planning for Manipulator Arms With Metric Temporal Logic Specifications," IEEE Robotics and Automation Letters, VOL. 3, NO. 1, JANUARY 2018.

[32] R.S. Johansson and J.R. Flanagan, "Coding and use of tactile signals from the fingertips in object manipulation tasks," Nature Rev. Neurosci.,vol. 10, no. 5, pp. 345–359, 2009.

[33] M.A. Abd, I.J. Gonzalez, T.C. Colestock, B.A. Kent, and E.D. Engeberg, "Direction of Slip Detection for Adaptive Grasp Force Control with a Dexterous Robotic Hand," IEEE/ASME International Conference on Advanced Intelligent Mechatronics, 2018.

[34] A. Prach, O. Kouba, and D. S. Bernstein, "How Slippery Is Viscous Friction," IEEE Control Systems Magazine, 2019, pp. 73-82.

[35] M. Quigley, et al, "ROS: an open-source Robot Operating System," ICRA Workshop on Open Source Software, 2009.

[36] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-backward error: Automatic detection of tracking failures", International Conference on Pattern Recognition , pages 2756–2759. IEEE, 2010.

[37] Y. Jonetzko, "Tactile based grasping with the biomimetic sensors BioTac and the Shadow Dexterous Hand", MS thesis, University of Hamburg, 2018.

[38] S. Zhe, et.al, "Force Estimation and Slip Detection/Classification for Grip Control using a Biomimetic Tactile Sensor", 2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids), 2015.

[39] N. Glossas and N. Aspragathos,"Fuzzy logic grasp control using tactile sensors," Mechatronics, vol. 11, no. 7, pp. 899–920, 2001.

[40] H. Yussof, M. Ohka, H. Suzuki, and N. Morisawa, "Tactile sensing-based control algorithm for real-time grasp synthesis in object manipulation tasks of humanoid robot fingers," IEEE International Symposium on Robot and Human Interactive Communication, 2008, pp. 377–382.

[41] J.M. Romano, K. Hsiao, G. Niemeyer, S. Chitta, and K.J. Kuchenbecker, "Human-inspired robotic grasp control with tactile sensing," IEEE Transactions on Robotics, vol. 27, no. 6, pp. 1067–1079, 2011

[42] M.O. Toole, K. Bouazza-Marouf, D. Kerr, and M. Vloeberghs, "Robust contact force controller for slip prevention in a robotic gripper,"Proceedings of the Institution of Mechanical Engineers, vol. 224, no. 3, pp. 275–288, 2010.

[43] E.D. Engeberg and S.G. Meek, "Adaptive sliding mode control for prosthetic hands to simultaneously prevent slip and minimize deformation of grasped objects," IEEE/ASME Trans. Mechatronics, vol. 18, no. 1, pp. 376–384, 2013.

[44] S.Michael. "Understanding Robust and Exploratory Data Analysis," Journal of the Royal Statistical Society: Series D (The Statistician) 320-321, 1984

[45] N. Wettels, J.A. Fishel, and G.E. Loeb, "Multimodal tactile sensor," The Human Hand as an Inspiration for Robot Hand Development. Springer, Cham, 405-429, 2014.

[46] N. Wettels, A.R. Parnandi, J. Moon, G.E. Loeb, and G.S. Sukhatme, "Grip Control Using Biomimetic Tactile Sensing Systems," IEEE Transactions on Mechatronics, vol. 14, no. 6, pp. 718-722, 2009.

[47] K. Wyk, J. Falco, "Slip Detection: Analysis and Calibration of Univariate Tactile Signals," arxiv 2018, `https://arxiv.org/pdf/1806.10451.pdf`.

[48] BioTac SP product manual. `https://www.syntouchinc.com/wp-content/uploads/2017/01/BioTac_Product_Manual.pdf`

[49] S. Smith, J. Tumova, C. Belta, D. Rus, "Optimal path planning for surveillance with temporal-logic constraints," The International Journal of Robotics Research, 2011.

[50] C. Baier, J.P. Katoen, K.G. Larsen, "Principles of Model Checking," MIT press, 2008.

[51] R. Alur, "Principles of Cyber-Physical Systems," MIT press, 2015.

[52] A. Bauer, M. Leucker, C. Schallhart, "Runtime verification for LTL and TLTL," Technical Report TUM-I0724, TU Munchen, 2007.

[53] W. Powell, "What You Should Know About Approximate Dynamic Programming," Naval Research Logistics (NRL), 56(3), 239-249, 2009.

[54] A. George, W.B. Powell and S.R. Kulkarni, "Value Function Approximation using Multiple Aggregation for Multiattribute Resource Management,"Journal of Machine Learning Research, 2008.

[55] S. Safra, "On the complexity of omega-automata," Annual IEEE Symposium on Foundations of Computer Science, 1988.

[56] N. Piterman, "From nondeterministic Buchi and Streett automata to deterministic parity automata," Annual IEEE Symposium on Logic in Computer Science, 2006.

[57] E.M. Wolff, U. Topcu and R.M. Murray, "Robust control of uncertain Markov decision processes with temporal logic specifications," IEEE 51st IEEE Conference on Decision and Control, 2012.

[58] R. Alur, S. Torre, "Deterministic generators and games for LTL fragments," TOCL, 2004.

[59] M. Alshiekh, R. Bloem, R. Ehlers, et al, "Safe reinforcement learning via shielding," arXiv preprint arXiv:1708.08611, 2017.

[60] P. Das, S.C. Mandhata, H.S. Behera, and S.N Patro, "An Improved Q-learning Algorithm for Path-Planning of a Mobile Robot," International Journal of Computer Applications (0975 – 8887) Volume 51– No.9, 2012.

[61] A. Konar, L. Jain and A. Nagar, "A Deterministic Improved Q-Learning for Path Planning of a Mobile Robot," IEEE Transactions on Systems, Man and Cybernetics: Systems, Vol. 43, No.5, 2013.

[62] M. Mostafa, and B. Bonakdarpour, "Decentralized runtime verification of LTL specifications in distributed systems," IEEE International Parallel and Distributed Processing Symposium, 2015.

[63] LTL3 tools, `http://ltl3tools.sourceforge.net/`

[64] Bauer, Andreas, Martin Leucker, and Christian Schallhart. "Comparing LTL semantics for runtime verification." Journal of Logic and Computation 20.3 (2010): 651-674.

[65] M. Hasanbeig, A. Abate, D. Kroening, Logically-Constrained Reinforcement Learning, https://arxiv.org/pdf/1801.08099.pdf, 2019.

[66] T. Icarte, Rodrigo, et al. "Teaching multiple tasks to an RL agent using LTL." Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[67] L. Xiao, C. Vasile, and C. Belta. "Reinforcement learning with temporal logic rewards." 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2017.

[68] Aksaray, Derya, et al. "Q-learning for robust satisfaction of signal temporal logic specifications." 2016 IEEE 55th Conference on Decision and Control (CDC). IEEE, 2016.

[69] J. Karlsson. Learning to Solve Multiple Goals. PhD thesis, University of Rochester, 1997.

[70] W.D. Smart, and L.D. Pack Kaelbling. "Effective reinforcement learning for mobile robots." In Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292), vol. 4, pp. 3404-3410. IEEE, 2002.

[71] P. Gastin, and D. Oddoux. "Fast LTL to Büchi automata translation." In International Conference on Computer Aided Verification, pp. 53-65. Springer, Berlin, Heidelberg, 2001.

[72] Li, Xiao, Yao Ma, and Calin Belta. "A policy search method for temporal logic specified reinforcement learning tasks." American Control Conference, 2018.

[73] G.D. Giacomo, L. Iocchi, M. Favorito, and F. Patrizi. "Reinforcement Learning for LTLf/LDLf Goals." arXiv preprint arXiv:1807.06333, 2018.

[74] Z. Xu and U. Topcu. "Transfer of Temporal Logic Formulas in Reinforcement Learning." arXiv preprint arXiv:1909.04256, 2019.

[75] G.D. Giacomo, M.Y. Vardi, "Linear Temporal Logic and Linear Dynamic Logic on Finite Traces," In Twenty-Third International Joint Conference on Artificial Intelligence. 2013.

[76] D. Giannakopoulou and K. Havelund. "Automata-based verification of temporal properties on running programs. " In ASE, 412–416. IEEE Computer Society, 2001.

[77] C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D.V. Campenhout. "Reasoning with temporal logic on truncated paths." In Conference on Computer Aided Verification. Warren A. Hunt Jr. and Fabio Somenzi, (ed.), Vol. 2725 of Lecture Notes in Computer Science, pp.27–39. 2003

[78] L. Lamport, "The temporal logic of actions." ACM Transactions on Programming Languages and Systems (TOPLAS), 1994.

[79] R. Vasumathi, et al. "Model predictive control with signal temporal logic specifications." 53rd IEEE Conference on Decision and Control. IEEE, 2014.

[80] T. Akazaki and H. Ichiro, "Time robustness in MTL and expressivity in hybrid system falsification." In International Conference on Computer Aided Verification, pp. 356-374., 2015.

[81] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals." Theoretical Computer Science 410.42, 2009.

[82] A. Donze, and O. Maler, "Robust satisfaction of temporal logic over real-valued signals." In International Conference on Formal Modeling and Analysis of Timed Systems, pp. 92-106, 2010.

[83] L. Lindemann and D. V. Dimarogonas. "Robust motion planning employing signal temporal logic." 2017 American Control Conference (ACC). IEEE, 2017.

[84] E. Plaku, and S. Karaman, "Motion planning with temporal-logic specifications: Progress and challenges." AI communications, 29(1), 151-162, 2016.

[85] L. Lindemann, D. Maity, J. S. Baras, and D. V. Dimarogonas, "Event-triggered Feedback Control for Signal Temporal Logic Tasks." In 2018 IEEE Conference on Decision and Control (CDC), pp. 146-151. IEEE, 2018.