Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2016-016 December 14, 2016

Propositional and Activity Monitoring Using Qualitative Spatial Reasoning

Spencer Dale Lane

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Propositional and Activity Monitoring Using Qualitative Spatial Reasoning

by

Spencer Dale Lane

B.S., Old Dominion University (2013)

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

Author .....
Department of Aeronautics and Astronautics
May 23, 2016

Certified by .....
Brian C. Williams
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Paulo C. Lozano
Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Propositional and Activity Monitoring Using Qualitative Spatial Reasoning

by

Spencer Dale Lane

Submitted to the Department of Aeronautics and Astronautics
on May 23, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Communication is the key to effective teamwork regardless of whether the team members are humans or machines. Much of the communication that makes human teams so effective is non-verbal; they are able to recognize the actions that the other team members are performing and take their own actions in order to assist. A robotic team member should be able to make the same inferences, observing the state of the environment and inferring what actions are being taken.

In this thesis I introduce a novel approach to the combined problem of activity recognition and propositional monitoring. This approach breaks down the problem into smaller sub-tasks. First, the raw sensor input is parsed into simple, easy to understand primitive semantic relationships known as qualitative spatial relations (QSRs). These primitives are then combined to estimate the state of the world in the same language used by most planners, planning domain definition language (PDDL) propositions. Both the primitives and propositions are combined to infer the status of the actions that the human is taking. I describe an algorithm for solving each of these smaller problems and describe the modeling process for a variety of tasks from an abstracted electronic component assembly (ECA) scenario. I implemented this scenario on a robotic testbed and collected data of a human performing the example actions.

Thesis Supervisor: Brian C. Williams
Title: Professor of Aeronautics and Astronautics

Acknowledgments

I would like to start by thanking my family and friends who were there for me through every step of this process. I would especially like to thank those who put up with my bad writing and even worse jokes. You know who you are.

I would also like to thank my fellow graduate students in the MERS lab as well as the post-docs. You all provided an excellent sounding board and helped point me in the right direction. You made my time in this lab much more enjoyable. Additionally, I would like to thank Andreas Hoffman for his advice and insight throughout my graduate career. Finally, I would like to thank my supervisor, Brian Williams. Without his guidance, none of this would have been possible.

This work was partially funded by a grant from the Mitsubishi Electric Corporation.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

THIS PAGE INTENTIONALLY LEFT BLANK

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

Processes in manufacturing tend to involve either groups of skilled workers or groups of machines. Even when more advanced robots are used on an assembly line, they tend to be cordoned off from humans. In recent years, there has been a push to facilitate teams of men and machines working in tandem. Communication between the two agents is key to ensure that these teams are effective. The robot needs to be able to observe the environment and the human, infer what is going on and how the current state reflects the human's goals, and choose actions that advance these goals. In human teams, a large amount of this communication is non-verbal. One person sees that the other is reaching towards a hammer and infers that they are going to pick it up. This thesis explores ways to replicate this non-verbal communication.

1.1 Domain Example

We define a simple example manufacturing scenario that will be used throughout this work to illustrate various points. In this scenario, a human and robot work together to perform electronic component assembly (ECA). To complete this task, four components must be placed, cleaned and then soldered into place. Each component starts in its designated bin and must be placed at its designated target. Two tools are used to facilitate the assembly process: a cleaner and a solderer. These both start in their designated bins. We label the components generically as red, blue, yellow, and green

rather than specifying what part they represent. We label each of the bins and targets with the component or tool it is associated with.

Five actions are available to the team: *pick*, *place*, *clean*, *solder*, and *pass*. In the *pick* action, one of the agents picks up an object from a designated location. In the *place* action, one of the agents places the object it is currently holding at the designated location. In the *clean* action, one of the agents uses the cleaner tool to clean the specified location. In the *solder* action, the human uses the solder tool to solder the component in the specified location. We decided that the robot could not solder, in order to give the human a specialized job. Finally, in the *pass* action, the robot holds out an object for the human to take and the human collects it from them.

Using this domain, we can illustrate the kinds of inferences used in this thesis for monitoring propositions and activities. We observe that the hand and the cleaner are in the same area and are moving together and from this we infer that the hand is holding the cleaner. We also observe that no objects are overlapping the red target and from this we observe that the red target is accessible. Given that the hand is holding the cleaner and the red target is accessible, we infer that the conditions required for the clean action to start have been met, however we have not observed any additional behaviors that lead us to believe that the clean action is starting. Later we observe that the hand and the cleaner are now moving towards the red target and are close to it. This additional behavior in conjunction with the conditions being observed, leads us to infer that the human has started cleaning the red target.

1.2 Problem Overview

In this work, we define and develop the Logical Activity Recognition System (LCARS) which is designed to recognize human actions and determine the state of certain features of the environment based on continuous sensor data. We split this problem into two sub-problems:

- Estimate discrete primitives from continuous sensor data

- Estimate predicates and actions from discrete primitives

Discrete primitives refer to finite domain discrete relations between two objects such as "the block is above the table" or "the tool is in the target area" They are intended as relations that are simple to calculate given perfect information, such as whether two regions overlap, and are meant to be combined to build other, more complicated relationships. This broad definition allows for the creation of domain specific primitives but LCARS is primarily designed to operate on a general set of primitives known as qualitative spatial relations (QSRs). Discrete primitives and QSRs are discussed in more detail in Chapter 3.

Predicates and actions are defined by a planning domain definition language (PDDL) model [16]. A PDDL model consists of a domain file and a problem file. The domain file defines the set predicates and actions for your system. The problem file gives an initial set of grounded predicates and an intended goal state.

The predicates that LCARS uses must either be in the form of discrete primitives, second order predicates (which can be defined in terms combinations of discrete primitives), or they can be predicates that are the results of an action. An example second order proposition is `(holding cleaner hand)`. This proposition can be inferred by the state of the discrete primitives, specifically "the cleaner is partially overlapping the hand" and "the cleaner and the hand are stable" meaning that they are in the same area and moving together.

For the purposes of monitoring PDDL actions, we first impose the restriction that the actions we wish to monitor must be defined as durative actions as defined in PDDL 2.1 [16]. Version 2.1 was specifically selected as it was the first version of PDDL to include actions with durations and it is still supported by many off the shelf planners. There is nothing in LCARS that would prevent using a later version of PDDL. Predicates and actions are discussed in detail in Chapter 4.

1.3 Approach Overview

This section discusses the architecture of LCARS and then describes our approach to solving the two main estimation problems. It also introduces our approach to learning some of the models required for estimation.

1.3.1 Architecture

LCARS consists of two key pieces: the discrete primitive monitor and the predicate and action monitor. Each of these serves a different and necessary role in the overall LCARS system. The system architecture is shown in Figure 1-1. The discrete primitive monitor is responsible for estimating the state of the discrete primitives. It requires a model of each of the primitive relations that it is responsible for monitoring. These models take the form of probabilistic hybrid automata (PHAs) which are combined into one overall concurrent probabilistic hybrid automaton (cPHA). At run time, the discrete primitive monitor takes as input the continuous sensor data that serves as the observations for each of the discrete primitives and outputs the belief state for each of the discrete primitives. This belief state is then fed as input to the predicate and action monitor.

The predicate and action monitor takes the belief state of each of the discrete primitives as input and outputs the belief state of the propositions and actions for the particular scenario at hand. In order to do this, LCARS requires a PDDL domain file, a PDDL problem file and a timed probabilistic concurrent constraint automata (tPCCA) that represents the overall action and predicate structure. At run time, the predicate and activity monitor takes as input the belief state of each of the discrete primitives and the status of any actions it is not monitoring but are still relevent to the predicates and actions that it is monitoring. In the ECA scenario, these actions are the actions that the robot is taking. LCARS does not monitor their status directly but still needs to update its representation of any predicates that depend on those actions. The predicate and action monitor outputs the belief state of the propositions and actions. These are given to an external executive which also supplies the status

of the external actions. The external executive manages the execution of actions by the robotic partner and as such needs to be informed of the state of the world and of the state of the human actions.

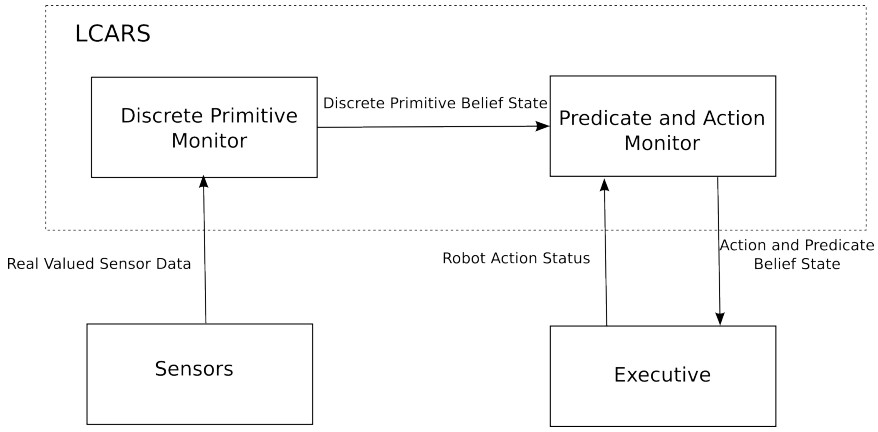


Figure 1-1: LCARS System Architecture

LCARS does not exist in a vacuum. It is designed to be integrated with a larger system and serves as the perceptual bridge between the human and the rest of the system. The predicate and action status is passed to an executive, which uses that information to dispatch actions on a robotic system that assist the human. One such executive is Pike [24]. Pike performs plan level monitoring and plan execution. It takes the proposition status as input, which LCARS can provide. It also issues commands to the robot and tracks the status of those actions, providing that information to LCARS.

1.3.2 Discrete Primitive Monitor

In order to intuitively model relationships between objects, we use discrete primitives. Discrete Primitives are intended to be simple, easy to express and easy to understand qualitative relationships between objects. For example, the hand is moving with a component or the hand is overlapping a component. These are built up into more complex relationships, specifically grounded propositions. As an example, when the hand is moving with an object and is overlapping it, it can be inferred that the hand is holding that object.

These primitives take the form of QSRs but we also allow for domain specific primitives to be defined. QSRs allow for an expressive ontology of semantic relationships that also have well defined rules for transition and commutation. The QSRs used in this thesis pull from the Region Connection Calculus (RCC) [14] and from the Qualitative Trajectory Calculus (QTC) [40]. Using the example above, the hand moving with the component becomes the hand and the component are *stable* and the hand is overlapping the component becomes the hand is *partially occluding* the component. We enhance this set by defining our own QSRs based on relative position and orientation.

We model primitives using a cPHA representation. cPHAs allow modeling both the continuous and discrete behavior of a system. In this case, the discrete portion consists of modes that represent the values that a particular primitive holds. For example, in the version of RCC used here, two objects can either be *discrete from* (DF), *partially occluding* (PO), *proper part of* (PPO) or *inverse proper part of* (iPPO) each other. Roughly, these correspond to the two regions being completely separate (DF), overlapping (PO), and one being completely contained within the other (PPO/iPPO). The cPHA model of an RCC relationship contains one discrete mode for each of these possible statuses. The continuous portion of the cPHA consists of the positions and orientations of each of the objects. Discrete primitives are discussed in more detail in Chapter 3.

1.3.3 Predicate and Action Monitor

In order to model predicates and actions, we use a model formalism called timed probabilistic concurrent constraint automata (tPCCA). tPCCAs are a class of models derived from work in probabilistic automata and timed automata [22]. They are a direct extension of probabilistic concurrent constraint automata (PCCA) [43, 44] with the added inclusion of clock variables. They are designed to allow a rich expression of durative actions and discrete predicates. tPCCA models are similar to cPHA models but differ in several ways. First, TCCA models do not allow for continuous variables. Second, they do not model the evolution of the state variables over time, with the

exception of a mode variable. Thirdly, they include a clock variable through which the mode transitions can depend on time.

Not all propositions can be directly observed through sensor readings. For instance, there is no way to directly infer whether a location is clean from the position of the cleaner and the targets. This contrasts with other propositions that can be inferred from the sensor data, such as whether or not a particular component is at a location. The status of the propositions that are not directly inferred from the sensors are instead inferred from the actions that have been executed. A location is considered to be clean after a clean action has been taken at that location.

We model the predicates by enumerating the grounded propositions that can be observed for a particular problem and indicating which of these can be observed and which are only the results of actions. We group those that can be observed into sets of propositions in which only one can hold at any time. For example, only one object can be at a particular location at one time and if no objects are at that location, it is considered empty. Because only one of the at predicates can hold at once, they would be grouped together. We then define one tPCA for each of these mutually exclusive sets with the mode variables in each case being one of the propositions. For those propositions that cannot be lumped into a group, we define a tPCA model where the discrete modes are *true* and *false*.

Actions are modeled using tPCAs as well with one tPCA per grounded action that LCARS is going to monitor. The tPCA for each of the actions is in fact time, with the timer being used to monitor the time bounds of the durative action. The modes in the action tPCA are used to indicate its status with the complete list of modes being: *ready*, *executing*, *finished*, *failed*, and *stopped*. The *ready*, *executing*, and *finished* are used to indicate the nominal stages of an action executing. The *failed* mode is used when certain failure conditions are met, notably when the time bound is exceeded. The *stopped* mode is triggered by the executive and is not normally used. The transitions in the action tPCA are based on the conditions defined in the PDDL domain and additional signaling behaviors. The signaling behaviors are what indicate that the action has actually started instead of it simply being able to start.

These are defined as a logical formula over predicates and discrete primitives. As an example, the clean action can when the hand is holding the cleaner and the target location is accessible but that alone does not indicate that it has actually started. Instead, the clean action starts when we observe that the hand and cleaner are moving towards and arriving at a particular target area. These are encoded as part of the guard condition on the transition between the ready and executing modes. Note that the construction of the tPCCA is currently performed manually though most of it could be learned and automated. Belief state update is then performed for the overall tPCCA that covers both the predicates and actions for a system.

1.3.4 Model Learning

In this thesis, we also show that it is possible to learn cPHA models with a models with a slightly limited formulation, allowing us to learn the discrete primitive models from data. We use an algorithm first introduced in [34] which learns guard conditions using multi-class classifiers from machine learning literature. Guard conditions describe regions within the continuous state space where a transition from one mode to another is likely. We found that the unsupervised methods presented in that work do not always work well in practice, particularly when the composition of the modes is important. We extend the method to several forms of supervised learning and use this to learn models for several of the discrete primitives used in this work.

1.4 Previous Work

1.4.1 Qualitative Spatial Reasoning

Qualitative spatial reasoning is a subfield of qualitative reasoning that specifically examines relations between objects in space. Much of the work has been on developing calculi that allow for rich representation and robust reasoning. This sort of reasoning has been applied in a wide range of application areas including Geographic Information Systems [12], biology [13] and robotic navigation [23, 25].

There are a large number of calculi that exist and many have different representation systems. Most of these focus on one particular aspect of spatial relations such as topology, direction, position, or shape. In this thesis, we wanted to use a set of calculi that together reflect several of these aspects and allow for reasoning in 3D space. We started with two of the more influential QSR sets, namely the Region Connection Calculus (RCC) [14] and Qualitative Trajectory Calculus (QTC) [40]. From there, we investigated representations for orientations and positions.

One of the earliest representations of relative orientation was developed by Schlieder [35]. This represents the relative orientation of line segments as clockwise, counter-clockwise and co-linear (parallel) and builds from there. This representation was extended into the STAR calculus which extended this to an arbitrary number of sectors [33]. Another representation is the Rectangle Algebra (RA) [2] which represents objects as bounding rectangles and describes relative rotations. Cardinal Direction Calculus (CDC) [19, 37] is similar to RA except that only the reference object is represented by a bounding rectangle. These two representations are complicated though, consisting of 169 and 511 basic relations respectively. All of these relations represent space as two dimensional so we could not use them directly.

Reasoning about QSRs has typically taken the form of a constraint satisfaction problem (CSP). A particular calculus provides a set of algebraic constraints and the current known state of the world can then be input and any hidden states inferred [4, 5]. In practice, it is necessary to assume that the knowledge of the world is imprecise and that the QSRs need to be estimated instead. This has been performed in a number of ways. RCC has been monitored using Hidden Markhov Models (HMMs) [38], probabilistic latent semantic analysis [3].

1.4.2 Activity Recognition and Propositional Monitoring

There are a wide variety of approaches to activity recognition and propositional monitoring. On the activity recognition side, there are many approaches to go from sensor data to activity recognition. These include using simple machine learning classifiers [32, 42], Bayesian networks [46], and recognizing temporal patterns [29, 45]. Another

recent approach combines the temporal pattern recognition and machine learning approaches and achieves more accurate recognition than either of those alone [26]. However, all of these approaches go directly from sensor inputs to activity status and do not utilize any of the plan level knowledge available based on what actions were available to execute at any given time.

A subfield of activity monitoring, known as workflow monitoring, uses sensor data to estimate what action is being performed, but knowledge of the sequence is also incorporated. Pody et al. use hierarchical-HMMs for monitoring operating rooms [28]. Pinhanez and Bobick use Past-Now-Future networks [31] which use Allen's temporal relations [1] to express ordering and allowed parallelism between events. Behera, Cohn, and Hogg combine QSR monitoring with workflow monitoring, estimating the status of spatial relations, using those to estimate the status of simple events, and using those to estimate the status of more complex events. They use probabilistic latent semantic analysis (pLSA) to estimate the QSRs and HMMs to estimate the activities [3]. These approaches tend to enforce a somewhat strict sequence on the activities, limiting the order in which they can be performed.

1.5 Thesis Outline

The rest of this thesis is structured as follows. Chapter 2 provides formal definitions of the problem that LCARS is solving as well as the models that are used throughout the rest of the work. Chapter 3 discusses discrete relations in more detail. Chapter 4 discusses the implementation of propositional and activity monitoring. Chapter 5 discusses how some of the models used can be learned. Chapter 6 discusses the implementation of the test system, the testbed that was used to assess it, and the experimentation that was performed as well as presenting and discussing the results of the experiments. Finally, Chapter 7 summarizes the contributions of this work and presents avenues for future work.

Chapter 2

Problem Statement

In this chapter, we formally define the inputs and outputs to LCARS, as introduced in Section 1.3. We begin by giving a formal definition of the problem that LCARS is solving and then define the model types that LCARS takes as input.

2.1 Problem Definition

As illustrated in the system architecture shown in Figure 1-1, LCARS consists of two main parts: the discrete primitive monitor and the predicate and activity monitor. Let us define the behavior of these in turn.

2.1.1 Discrete Primitive Monitor

The discrete primitive monitor is designed to estimate the current status of the discrete primitives for a particular scenario. It operates on a cPHA model of the entire set of relations for a particular run. Formally, the problem that the discrete primitive monitor is designed to solve is as follows:

Definition 2.1 (Discrete Primitive Monitor Problem)**.** *Given a set of discrete primitives modeled as a cPHA $\mathcal{CA}$, an a priori distribution of the system state (continuous and discrete) $p(\mathbf{x})$, and the observations of the system $\mathbf{y}_c$, estimate the hybrid state of the system $\hat{\mathbf{x}}$.*

The cPHA formalism is defined in Section 2.2. In our case, the observations of the system are the continuous sensor readings. The specific sensor readings depend on the set of primitives being monitored.

2.1.2 Predicate and Activity Monitor

The predicate and activity monitor is designed to estimate the current status of the grounded PDDL propositions and the grounded actions. It operates on a tPCCA model of the set of propositions and actions that are possible for a particular PDDL domain and problem. Formally, the problem that the monitor is designed to solve is as follows:

Definition 2.2 (Predicate and Activity Monitor Problem)**.** *Given a set of propositions and actions defined as a tPCCA, an a priori distribution of the system state* $p(\mathbf{x^0})$*, the observations of the system* $y^{<0,t>}$*, and the commands from the executive* $\mu^{<0,t>}$*, iteratively calculate the belief state* $P(x_i^{t+1}|y^{<0,t>}, \mu^{<0,t>})$*.*

Where the state x_i is a full assignment to the mode variables $x_i \in \Sigma^m$, $y^{<0,t>}$ is the series of observations from time 0 to time t and $\mu^{<0,t>}$ is the series of commands from time 0 to time t. In this case, the modes are the predicate and action statuses, the observations are the discrete primitive statuses and the commands are the robot action statuses and stop commands from the executive.

2.2 Concurrent Probabilistic Hybrid Automata

2.2.1 PHA Formalism

A cPHA model is made up of several PHA models. Each PHA contains several discrete-time difference equations for each of several discrete modes. It also includes probabilistic transitions and constraints as to when those transitions can occur. The definition given here is based on the original definition by The definition of PHAs used in this work is the original definition by Hofbaur and Williams [21]. Formally, a probabilistic hybrid automaton A is defined as the tuple $A_a = \langle x_a, w_a, F_a, T_a, X_a^d, U_a^d, T_a^s \rangle$.

- x_a denotes the hybrid state, $x_a = \{x_a^d\} \cup x_a^c$. x_a^d is the finite domain discrete mode variable with the domain of X_a^d. x_c is the set of continuous state variables $x_a^c = \{x_a^{c1}, \ldots, x_a^{cn}\}$ with the domain R^n.

- w_a is the set of input output variables $w_a = u_a^d \cup u_a^c \cup y_a^c$. u_a^d refers to the discrete input variables $u_a^d = \{u_a^{d1}, \ldots, u_a^{dm_d}\}$ which has a domain of U_a^d. u_a^c refers to the set of continuous input variables $u_a^c = \{u_a^{c1}, \ldots, u_a^{cm_c}\}$ which has a domain of R^{m_c}. Finally, the continuous output variables are $y_a^c = \{y_a^{c1}, \ldots, y_a^{cm_y}\}$. This has a domain of R^{m_y}.

- $F_a : X_a^d \rightarrow F_a^{DE} \cup F_a^{AE}$ defines the continuous evolution in terms of discrete difference equations F_a^{DE} and algebraic equations F_a^{AE}. T_s indicates the sampling period for the equations.

- The finite set of transitions T_a models the probabilistic changes in the discrete mode. Each transition is written in the form of a tuple $\langle \tau_a^i, g_a^i \rangle \in T_a$. Each of these functions is associated with the guard condition g_a^i and a probability mass function over the modes.

The full concurrent probabilistic hybrid automaton is written as a set of PHAs. It also defines the noise function in the form of additive Gaussian processes. These disturbances are used to model both process noise and sensor noise.

2.2.2 cPHA Formalism

A cPHA is formally defined by the tuple $\langle A, u, y_c, S \rangle$.

- $A = \{A_1, A_2, \ldots A_n\}$ is the set of n PHAs that are contained within the overall cPHA.

- $u = u_d \cup u_c$ is the set of inputs and command variables respectively. Note that this is for the overall automaton and not each individual PHA. Because the PHAs can be interconnected, the output of one PHA might be the input of another PHA. Those variables are not included in this set.

- $y_c \subseteq y_c1 \cup y_c2 \cup \ldots \cup y_cn$ is the set of outputs of the overall cPHA.

- $S = \{s_1, s_2, \ldots, s_n\}$ is a series of constraints that indicate how the set of PHAs in A are connected.

2.3 Timed Probabilistic Concurrent Constraint Automata

2.3.1 tPCA Formalism

tPCCA models are made up of a set of timed probabilistic constraint automata (tPCA) operating concurrently. We first describe a single automaton and then the combination of multiple. A tPCA for a component "a" is defined by the tuple $A_a = \langle \Pi_a, M_a, T_a, P_{T_a} \rangle$:

- $\Pi_a = \Pi_a^m \cup \Pi_a^t \cup \Pi_a^r$ is a set of discrete variables describing component "a". Π_a^m is a singleton set containing the mode variable $x_a = \Pi_a^m$ whose domain $D(x_a)$ is the finite set of discrete modes for A_a. Π_a^t is the unique clock variable t_a for A_a whose domain $D(t_a)$ is the set of positive integers. Π_a^r is the set of attribute variables which includes any inputs, outputs, guard variables, control variables, and any other discrete variables that define the behavior of the component. The attribute variable set has a finite domain $D(\pi_a^r)$. Σ_a is the set of all partial assignments over Π_a which represents a full assignment to $\Pi_a^m \cup \Pi_a^r$ and the state space $\Sigma_a^{x_a} = \Sigma_a \Downarrow_{x_a}$ is the projection of Σ_a onto the mode variable x_a.

- $M_a : \sigma_a^{x_a} \rightarrow C(\Sigma_a^r)$ is a mapping of each mode assignment to a finite domain constraint $c(x_a = v_a) \in C(\Sigma_a^r, \Pi_a^t)$, where $C(\Pi_a^r, \Pi_a^t)$ is a set of constraints over $\Pi_a^t \cup \Pi_a^r$. These are known as modal constraints. These constraints are expressed in terms of propositions with equality, $\lambda ::= true | false | (l^0 = v) | (u = v) | (t \text{ op } r) | \neg\lambda_1 | \lambda_1 \wedge \lambda_2 | \lambda_1 \vee \lambda_2$. The allowed clock operations (t op r) are $c < r$, $c \leq r$, $c > r$ and $c \geq r$.

- $T_a : \Sigma_a^{x_a} \times C(\Pi_a^{x_a} \rightarrow \Sigma_a^{x_a})$ defines the set of transition functions. For each transition function $\tau_a \in T_a$ we define a guard condition $g_a \in C(\Sigma_a^r)$. The transition functions therefore specify a mode assignment $(x_a = v'_a) \in \Sigma_a^{x_a}$ that could be reached at time $t + 1$.

- $P_{T_a} : T_a(x_a = v_a, g_a) \rightarrow R[0, 1]$ is the transition probability distribution. It defines a probability distribution across all the transitions into the possible target modes. The target modes are defined by the transition functions $T_a(x_a = v_a, g_a)$. A probability distribution must be defined for each mode assignment and each guard.

2.3.2 tPCCA Formalism

An entire plant P is a set of tPCA models, formally defined by the tuple $P = \langle A, \Pi, Q \rangle$:

- $A = \{A_1, A_2, \ldots, A_n\}$ is the set of tPCA that make up the n components of the plant.

- $\Pi = \cup_{a=1\ldots n}\Pi_a$ is the set of all variables. The variables are partitioned into various smaller sets. These are mode variables $\Sigma^m = \cup_{a=1\ldots n}\Pi_a^m$, clock variables $\Sigma^m = \cup_{a=1\ldots n}\Pi_a^m$, control variables, $\Sigma^m = \cup_{a=1\ldots n}\Pi_a^m$, observation variables $\Sigma^m = \cup_{a=1\ldots n}\Pi_a^m$, and dependent variables $\Sigma^m = \cup_{a=1\ldots n}\Pi_a^m$. We also define a set of full assignments over the different types of variables Σ^u, Σ^o, and Σ^d mapping to Π^u, Π^o and Π^d respectively.

- $Q \subset C(\Pi)$ is the set of finite domain constraints that capture the interconnections between the various automata.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3

Monitoring Discrete Primitives

In the previous chapter, we examined the overall structure of LCARS and gave a formal definition of the various components. In this chapter, we give more detail on how the discrete primitives are modeled. Discrete primitives are simple discrete relationships between objects that express part of the state of the two objects in a semantically significant way. For instance "The cleaner is above the red target". These relationships primarily take the form of Qualitative Spatial Relations (QSRs), though other kinds of relationships could be integrated into LCARS.

QSRs are often, though far from always, modeled using HMMs [38]. We instead use a PHA representation. This allows us to directly map the continuous inequality constraints that define each discrete primitive to the transition guards of the PHA. We then perform belief state update over the entire cPHA to estimate the current status of each of the primitives.

This chapter defines the set of QSRs used in this thesis. Each of the QSRs are defined in terms of their continuous inequality constraints. The mapping of these inequality constraints to a PHA definition is then discussed. It also discusses the axis conventions used throughout this work.

3.1 Axis Conventions

In this work, we use the axis conventions and colorations used in the Robot Operating System (ROS) which is similar to the conventions used in other robotics sources. The X axis is defined as pointing forward and is always depicted in red. The Y axis is defined as pointing left and is always depicted in green. The Z axis, in order to maintain a right hand coordinate system, is defined as pointing up and is always depicted in blue. An example coordinate system is shown in Figure 3-1.

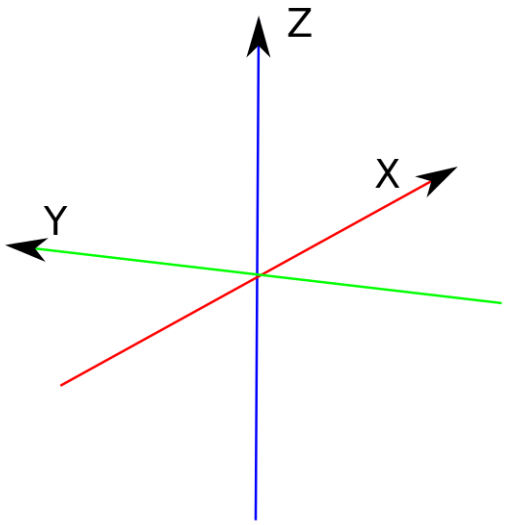


Figure 3-1: Labeled Axes

3.2 Qualitative Spatial Relations (QSRs)

Qualitative Spatial Relations (QSRs) are meant as a way to represent problems intuitively and reason about them in a manner that is intuitive to humans. When describing the location of an object in space, people do not give a precise numerical description but rather use qualitative relations such as "the chair is in front of the desk" or "the table is in the kitchen." Because humans naturally use these sorts of descriptions, it can be useful to reason on this level and recognize the current status of the various relations.

There are many different kinds of relations and many different formalisms for expressing them. It is important to be able to express a wide variety of different situations while still maintaining the ability to observe the relations and their semantic significance. Many of the calculi that have been developed have a rich set of relations but are not able to easily be described with words.

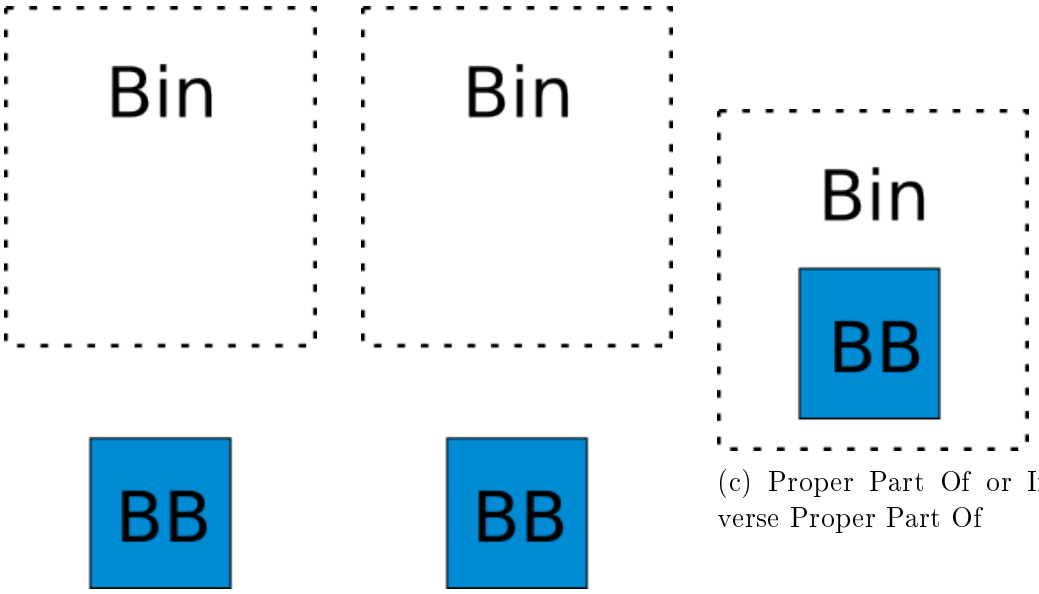
For this work, we have defined a set of qualitative spatial relations that we believe are able to be monitored and cover a wide range of possible applications. These are three-dimensional relations and fall under the broad categories of Cartesian relations and rotational relations. Cartesian relations deal with relative positions and rotational relations deal with relative orientations.

3.2.1 Cartesian Relations

Region Connection Calculus

Region Connection Calculus (RCC) is used to describe how two regions of space overlap. For this work we are using RCC5 [14] which contains 5 relations: *discrete* from (DF), *partially occluding* (PO), *proper part* (PP), *inverse proper* part (iPP), and *equals* (EQ). For the purposes of monitoring, PP and iPP are equivalent, as region a being a proper part of region b implies that region b is an inverse proper part of region a. They thus identical, monitoring for region a being a proper part of region b implies monitoring b being an inverse proper part of a. given the very specific nature of EQ and the relative difficulty of estimating it using noisy sensors, we have removed it. Excepting the trivial solution of region a equaling region b, two regions being equal requires a very specific set of circumstances. The two regions must be precisely the same size and shape, they must be in precisely the position, and their orientations must be aligned in some way. With noisy sensors, it is challenging to state with certainty that the positions and orientations of two regions match in that way. The set of relations that we are monitoring (DF, PO, PP, and iPP) are shown in Figure 3-2.

This can be defined as a set of inequality constraints over the relative distance.



(a) Discrete From (b) Partially Occluding (c) Proper Part Of or Inverse Proper Part Of

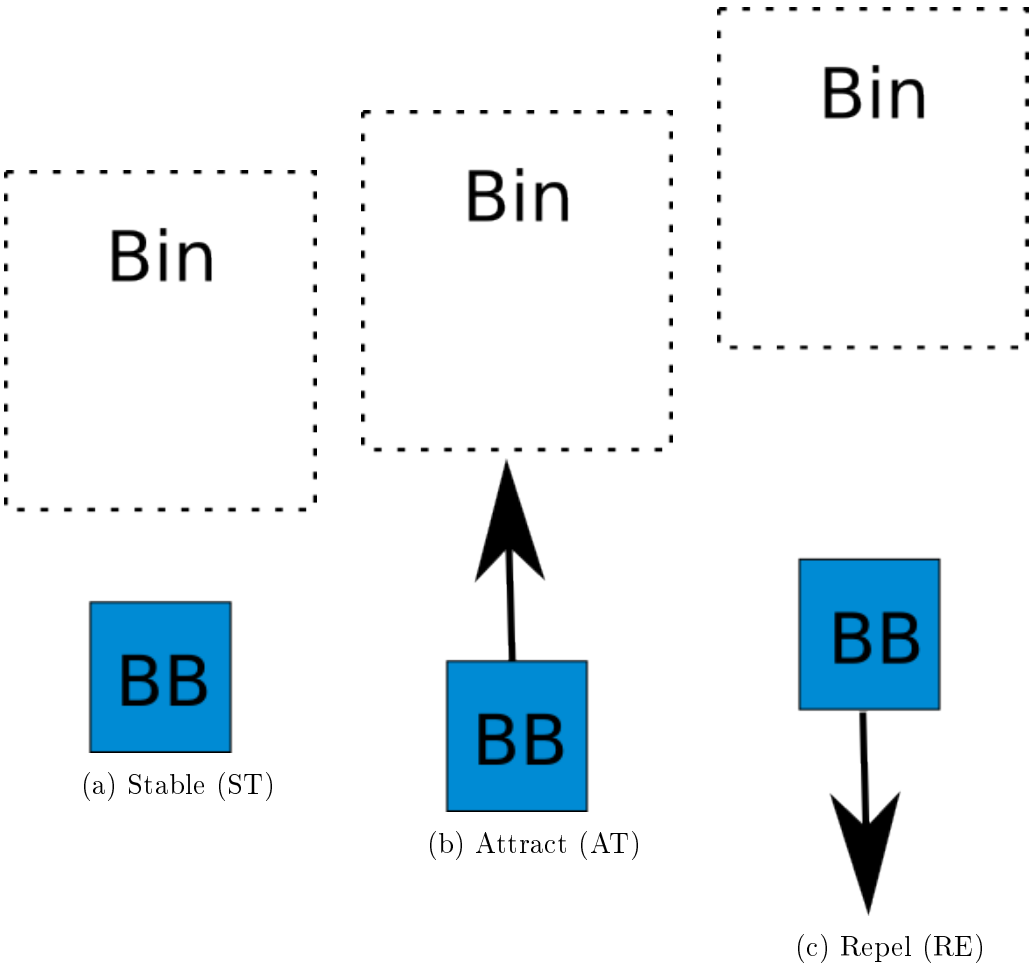
Figure 3-2: The three relations from Region Connection Calculus (RCC) that we are using

We can define a few key distances between the objects such that we result in this set of constraints. When the distance is larger than the first key distance, the two objects are discrete from each other. When the distance is between the two key distances, the two objects are partially overlapping. When the distance is less than the second key distance, the two regions are partially overlapping. Note that the actual distances depend on the orientation of the objects. This is why the probabilistic transitions are important in this formulation. Also note that some objects may not be able to be a proper part of the other. For example, the cleaner cannot be a proper part of the cleaner as the two are solid objects. In this case, we still define two key distances. The first is one at which they are likely partially occluding and the second is a distance at which they are almost certainly partially occluding.

Qualitative Trajectory Calculus

Qualitative Trajectory Calculus (QTC) is used to describe the relative motion of two points in space, such as moving together or apart [40]. It extends simply to three dimensions. In this work we use a subset of the relations: *attract* (AT), *repel* (RE)

and *stable* (ST). *Attract* corresponds to a decrease in the relative distance. *Repel* corresponds to an increase in the relative distance. *Stable* corresponds to a state where the relative distance remains the same. The set of relations that we are using is shown in Figure 3-3.



(a) Stable (ST)

(b) Attract (AT)

(c) Repel (RE)

Figure 3-3: The three relations from Qualitative Trajectory Calculus that we are using

Positional Relations

We also define several other relationships based on the relative position and orientation of two objects. The relations are: *in-front* or *behind*, which correspond to the x-axis; *left* or *right*, which correspond to the y-axis; and *above* and *below*, which correspond to the z-axis. These conventions are shown in Figure 3-4 and an example

is shown in Figure 3-5. These relations are intended to be written semantically as "Foo is {left,right} and {above,below} bar, relative to baz." This defines a coordinate system whose origin is centered at bar and whose orientation is that of baz. If no relative frame is defined, the coordinate system has the same orientation as bar.

This allows for relations to be written with a variety of coordinate systems. For example, "The red component is in-front of the target, relative to the robot" is very descriptive of where the red component is in space.

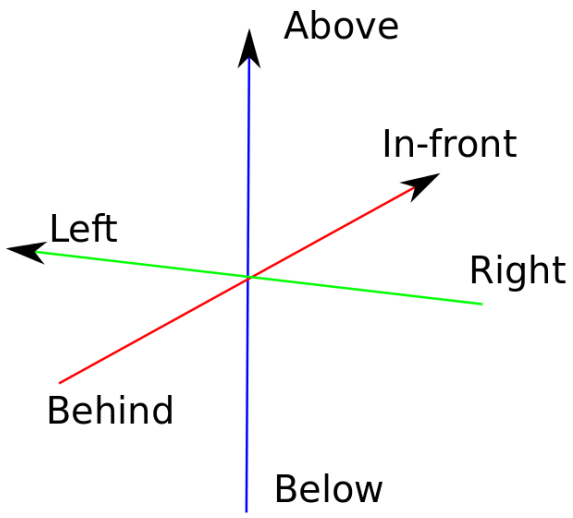


Figure 3-4: Relative Position Relations. Forward and backward correspond to the x-axis; left and right correspond to the y axis; above and below correspond to the z-axis

These can be expressed through a series of inequality constraints. The continuous variable is the position of the first object in the reference frame defined by the rest of the relationship. The constraints are then based on the sign of the particular coordinate. For example, if the x coordinate is positive, then the object is *in-front* and if it is negative or zero, it is *behind*. Likewise we can map positive to *left* and above on the y and z axes respectively. We can also map *right* and *below* to negative or zero in the y and z coordinates respectively.

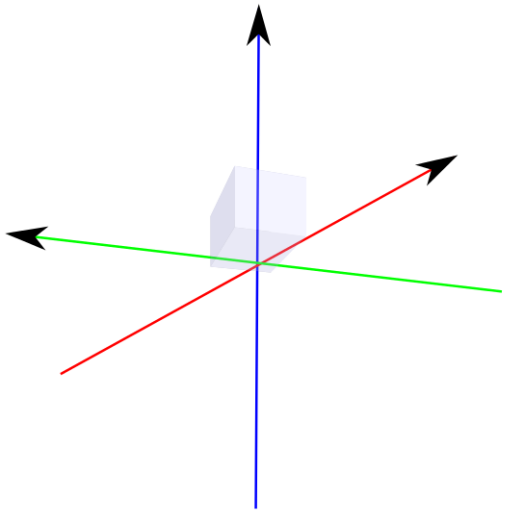
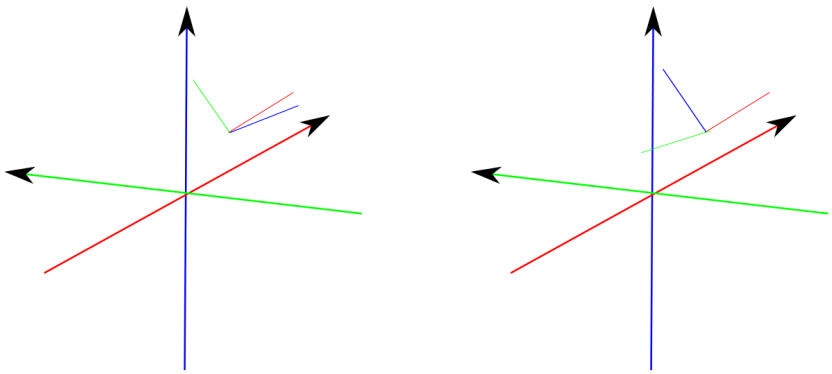


Figure 3-5: An example of the position relationships, the cube is in-front of, left of and above the reference

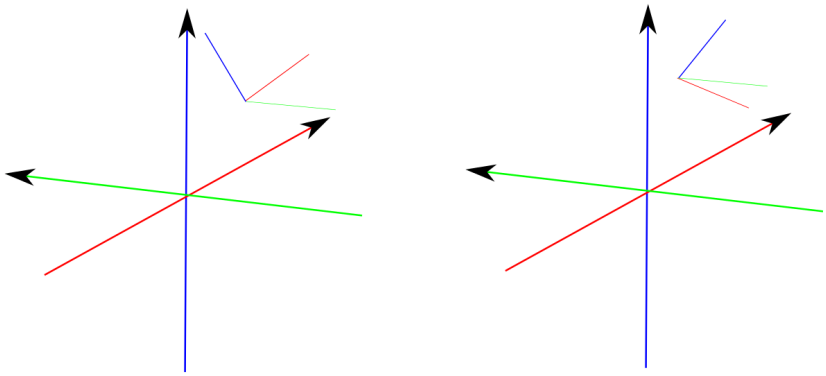
3.2.2 Rotational Relations

We also define a set of relationships based on the relative rotations of two objects. In each of the axes, we define several relations: *aligned*, *anti-aligned*, *perpendicular clockwise*, *perpendicular counter clockwise*, *rotated clockwise* and *rotated counter clockwise*. Each of these relations is defined for rotations around each of the three axes with rotations around the x, y, and z axis being known as roll, pitch, and yaw respectively.



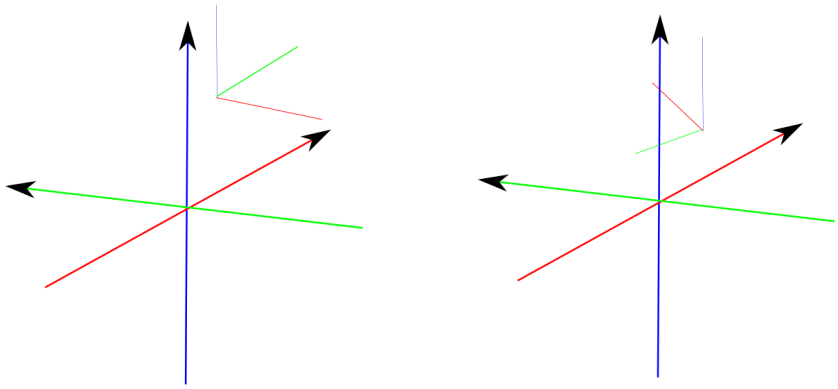
(a) Rolled Clockwise (b) Rolled Counter Clockwise

Figure 3-6: Roll rotation relations



(a) Pitched Clockwise (b) Pitched Counter Clockwise

Figure 3-7: Pitch rotation relations



(a) Yawed Clockwise (b) Yawed Counter Clockwise

Figure 3-8: Yaw rotation relations

We can define these relations as inequality constraints over the angles of rotation. We define our continuous variables as the Euler angles representing roll, pitch and yaw. We limit these to $[-\pi, \pi)$ and wrap when those bounds are exceeded. The *aligned*, *anti-aligned* and the two *perpendicular* relations are most obviously defined as equality constraints, however, strict equality constraints are not conducive to accurate monitoring. We therefore relax the equality constraints and define the inequality constraints in each axis as:

$$c = \begin{cases} \text{aligned, if } -\epsilon < \theta < \epsilon \\ \text{anti-aligned, if } \pi - \epsilon < \theta \text{ and } \theta < -\pi + \epsilon \\ \text{perpendicular-clockwise, if } \pi/2 - \epsilon < \theta < \pi/2 + \epsilon \\ \text{perpendicular-counter-clockwise, if } -\pi/2 - \epsilon < \theta < -\pi/2 + \epsilon \\ \text{rotated-clockwise, if } \epsilon < \theta < \pi/2 - \epsilon \text{ and } \pi/2 + \epsilon < \theta < \pi - \epsilon \\ \text{rotated-counter-clockwise, if } -\pi/2 + \epsilon < \theta < -\epsilon \text{ and } -\pi/2 + \epsilon < \theta < -\pi/2 - \epsilon \end{cases} \tag{3.1}$$

Where c is the active constraint, θ is the angle of rotation in that axis, and ϵ is the relaxation factor.

3.2.3 Modeling QSRs as PHAs

Let us now examine how we map the series of inequality constraints to a PHA. First, we will look at the RCC relations. In this PHA, the continuous state variable is used to represent relative distance and the guard dynamics describe the magnitude of the relative distance. These guards depend on the pair of objects being compared but roughly correspond to the blocks being "far," "near," "very near," and "almost identical." The relative distances are measured from the center of the regions being compared. The transition structure of this model is shown in Figure 3-9. This is used to illustrate the most likely transitions given the guards being active.

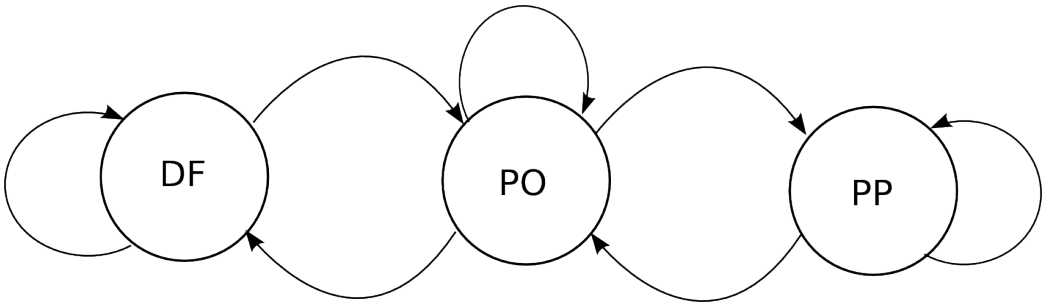


Figure 3-9: PHA model of RCC5

QTC is modeled using a similar method. The continuous state variable is the rate

of change of the relative distance between the two objects. The guard functions are based on the sign of the continuous state. The structure of the most likely transitions based on the current guard conditions is shown in Figure 3-10.

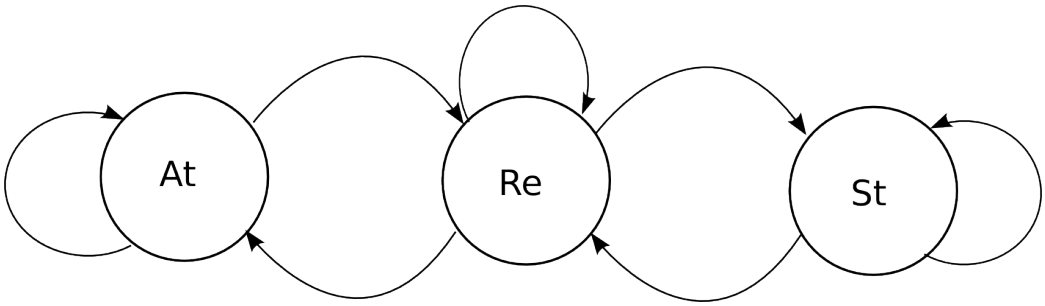


Figure 3-10: PHA model of QTC relations

While we are not expecting a significant performance increase from switching from HMMs to PHAs, PHAs do provide a significant advantage in terms of model expressiveness. They allow us to encode hard and soft state constraints in a relatively natural manner. In particular, each of the calculi discussed here defines various algebraic constraints. For example, the positional calculus defined in Section 3.2.1 and given the same reference frame, if A is above B and B is above C, then we can infer that A is above C. Similarly in RCC, if A is a proper part of B and B is a proper part of C, then A must be a proper part of C. While these algebraic rules can be encoded as part of the observation function within an HMM, it is natural to encode them as part of the transition guards within a cPHA. This application of a cPHA is not yet implemented as part of LCARS but is a large space for potential improvement. This would bring the PHA approach in line with other approaches to monitoring QSRs such as latent semantic analysis which encodes these constraints as part of logical programs [3].

3.3 Other Discrete Primitives

The formulation of discrete primitives as PHAs also allows for defining domain specific discrete primitives. In general, domain specific primitives are used to describe behavior that is important to a domain that cannot be written in terms of the QSRs

described here. The exact format of a domain specific primitive will depend on the specific behavior or relation being represented. In order for LCARS to monitor one of these primitives, a PHA model of it must be provided by the user. We will discuss the hand-off task that is part of the *pass* action in the ECA scenario.

The *pass* action involves the robot handing an object to a human by extending its arm towards the human and releasing the object when it detects that the human is holding it. If we simply had the robot release the object when it observed that the human was holding the object, there is a chance that the human will not have a very firm grip and will instead drop the object. If the object that was being handed off was delicate, this could be disastrous and regardless, an object being dropped lowers team efficiency. Instead, we define a new discrete primitive with two states *gripped* and *not gripped*. The use of PHAs for this purpose was previously demonstrated by Lars Blackmore and Steve Block [9].

We learned the PHA model for this primitive using the method discussed in Chapter 5. We examined the angle and angular velocity of the wrist joint of the robot during the hand-off task and observed a sharp difference between the gripped and not gripped states. Using supervised learning, we were able to learn a PHA model that allowed us to differentiate between *gripped* and *not gripped* with the arm at a particular configuration. This model worked for all of the objects we tested without making any modifications.

Other domain specific discrete primitives could be either modeled by hand or learned as this one was. Modeling relations by hand is time consuming but accurate. The limitations of the currently implemented model learning algorithm are discussed in detail in Chapter 5 but the primary drawback is that it is currently limited to a PHA model with a specific formulation which requires the continuous dynamics to be linear. This can be overcome in the long run but it limits what can be learned at present.

3.4 Chapter Summary

In this chapter we presented a strategy for modeling and monitoring discrete primitives. We took the continuous inequality constraints imposed by a particular QSR and mapped that to the transitions within a PHA. One PHA is defined for each pairwise relation for a particular model and at run time belief state update is run on the overall system of PHAs, in order to estimate the probability of each of these holding true.

We described a set of QSRs that are useful for modeling a wide variety of relations between two objects in 3D space. We started with the existing RCC and QTC calculi and supplemented them with additional relationships, relating to relative position and orientation. We first described the inequality constraints for each of these calculi and then described how to map these onto a PHA. We also described a process for adding additional discrete primitives and used the pass action as an example.

In the next chapter we describe the monitoring of PDDL predicates and actions, completing the detailed description of the core components of LCARS.

Chapter 4

Monitoring Actions and Predicates Expressed in PDDL

The Planning Domain Definition Language (PDDL) was developed as a standard language for defining planning problems and is widely used across the planning community. It has gone through many incarnations and is currently on version 3.1. In this thesis, we work with PDDL 2.1 [16].

Similarly to monitoring the discrete primitives, the strategy for monitoring predicates and actions is to map each to a probabilistic transition system, specifically a tPCCA. The predicates are grouped into sets where only one proposition may hold at a time. The transitions in those models are conditioned on the belief state of the discrete primitives. The actions are set up so that there is one tPCA per possible action that could occur for a particular PDDL problem. The timed component is used to monitor the duration of the durative actions. An action cannot finish until the lower bound on time is met and an action has failed if the upper bound has been violated. The transitions in the action models are conditioned on the belief state of the predicates that appear in that actions conditions, effects and additional signaling behaviors. The automota for each of the predicates and actions are combined into one overal tPCCA. Belief state update is then performed over that concurrent automaton.

In this chapter, we discuss PDDL representations, show an example as applied to the electronic component assembly problem and then talk about converting PDDL

propositions and actions into tPCCA format.

4.1 PDDL Representations

PDDL Models are split into two parts, the domain and the problem. Domain files define the types of predicates and actions that can occur for a particular class of problem, and problem files define a start state and goal for a particular instance of a domain.

In this work, we only consider predicates and domains that do not contain numeric fluents. LCARS could in principle be extended to include numeric fluents, PHA models would be well suited to this task, however that is outside the scope of this work. We also only consider durative actions. Other actions in PDDL are atomic and instantaneous. Because they do not have any duration, atomic actions are not well suited to the action structure defined here. Again, a modification to LCARS to include atomic actions could be devised but it is outside the scope of this work.

4.1.1 Predicates and Propositions

The predicates for a particular domain are defined at the begining of the domain file. Each predicate takes arguments which can be generic or typed. A predicate with these arguments filled in is known as a proposition or a grounded proposition. Each proposition has an associated truth assignment for each point in time. A proposition's mapping from time to truth assignment is called a fluent.

To further elaborate, let us examine the holding predicate which is defined as: `(holding ?obj - object ?manip - manipulator)`. The `holding` predicate takes two arguments, `obj` and `manip`. Both of the arguments are typed with their types being `object` and `manipulator` respectively. A grounded proposition could be `(holding redcomponent hand)` which corresponds to the hand holding the red component.

4.1.2 Durative Actions

PDDL Durative Actions are an extension of basic PDDL actions such that they take some amount of time to occur. Basic PDDL actions simply have parameters, preconditions, and effects, where the preconditions and effects are defined as logical combinations of predicates, i.e. propositional formulae, written in terms of the parameters. Durative actions simply add a duration and modify the format of the preconditions and effects.

Preconditions or, in the case of durative actions, conditions are requirements that must be met at certain stages throughout an action. For instance, in order for the hand to pick up an object, the object must not be held by anything and the hand must not be holding anything. Effects are simply the results of the different stages of an action. For example, when the solder action finishes, the object that was being soldered is now soldered into place.

Each predicate within the conditions of a durative action has a temporal annotation. These are used to indicate when the condition is in effect and can take one of three forms: (at start (*predicate*)), (at end (*predicate*)), or (over all (*predicate*)). The *at start* conditions must hold at the beginning of the action and the *at end* conditions must hold at the end of the action. For example, the hand must be holding the cleaner in order for the clean action to start and both the human must have a firm grip on the object in order for the pass action to complete. The *over all* conditions on the other hand must hold from the time immediately following the start of an action until the point immediately preceding the end of the action. For example, the cleaner must be held throughout the clean action. If for some reason the cleaner is no longer held, something has gone wrong. Overall conditions are also sometimes known as invariant conditions.

The effects of an action are likewise temporally annotated. The predicates that are applied as part of effects are labeled as either (at start (*predicate*)) or (at end (*predicate*)). As an example, the location being cleaned will become blocked at the start of the clean action and the hand will be holding an object at the end of a pick

action. It is worth noting that effects applied at the start of an action can achieve an overall condition and effects applied at the end of an action and violate an overall condition.

The duration of durative actions is expressed as equality constraints or as inequalities. In the case of monitoring, it is preferable to have inequalities, as in any actual application, it is very unlikely for an action to take an exact and specific amount of time. Instead, we want to define a lower and upper bound for the duration of each action. As an example of a duration, the clean action takes at minimum 3 seconds, because that is how long it takes to thoroughly clean a location, and at most 15 seconds. If the action goes any longer than 15 seconds, something has gone wrong.

4.2 Electronic Component Assembly Example

For illustrative purposes, let us examine part of the ECA domain definition:

```
(empty ?manip - manipulator)
(holding ?obj - object ?manip - manipulator)
(iscleaner ?obj - object)
(reachable ?loc - location ?manip - manipulator)
(isclean ?loc - location)
(:durative-action clean
  :parameters  (?loc - location ?manip - manipulator ?obj - object)
  :duration :duration (and (< ?duration 15) (> ?duration 3)
  :condition (and
            (at start (reachable ?loc ?manip))
            (at start (holding ?obj ?manip))
            (at start (isCleaner ?obj))
            (over all (holding ?obj ?manip)))
  :effect (and
            (at end (isclean ?loc)))
)
```

`(iscleaner ?obj - object)` exists only to label one of the objects, specifically a tool, as being the cleaner object. It is defined as an initial condition within the problem file and cannot change during execution, and for the purposes of estimation, it cannot be sensed.

`(reachable ?loc - location ?manip - manipulator)` is used to indicate whether a particular location is reachable by a particular manipulator. This is also set in the problem file and cannot change during runtime. The human is assumed to be able to reach all of the locations, it is primarily used for the robot which cannot reach all locations with both of its manipulators.

`(isclean ?loc - location)` is only used as an effect of the clean action. It cannot be sensed and is exclusively a result of the clean action finishing.

This leaves `(empty ?manip - manipulator)` and `(holding ?obj - object ?manip - manipulator)` which change during runtime and can be estimated. We will use these predicates as examples in Section 4.3.1.

4.3 Constructing tPCCA Models from PDDL Representations

We build off of the work of David Wang who converted PDDL representations to Timed Constraint Automota (TCA) representations for the purposes of solving planning problems [41]. We augment Wang's TCA encoding by adding structures to facilitate recognition and by adding control variables to the actions to facilitate resetting during the monitoring process, allowing the system to restart after an action has failed. In this section, we discuss the construction of tPCCA models from the PDDL representation for a particular problem. We discuss both propositions and durative actions.

4.3.1 Propositions

When defining the set of proposition models for a particular problem, the list of predicates must first be split into two groups, those that are simply the results of actions, and those that can be monitored from sensing input. The split is done to separate predicates that can be monitored by LCARS from predicates that cannot. Note that the structure of two propositions with the same predicate but different arguments will have the same structure with different variables. Proposition models do not require clock variables and are therefore simply PCA.

Let us use the `(isclean ?loc - location)` predicate as an example of the first group. This action is the result of a clean action being performed and for a particular location can only be achieved by one of three actions:

- `(clean ?loc hand cleaner)`
- `(clean ?loc baxterleft cleaner)`
- `(clean ?loc bexterright cleaner)`

Of these actions, only `(clean ?loc human cleaner)` is monitored directly by LCARS; the other two actions are performed by the robot and are thus handled by an external executive. In order to model the state the isclean predicate, we define a PCA with two states: *true* and *false*. We also define three attribute variables that correspond to the state of the three actions that can achieve this predicate. The state of the action performed by the human is read from the tPCA model defined for it in LCARS. The state of the robot actions must be passed in from an external executive. Three transitions are then defined. The first one has a guard condition corresponding to one of the three actions having finished. This causes a transition from *false* to *true* with 100% probability. The two are a self transition from *false* to *false* if that condition is not met and a self transition from *true* to *true* in all cases. This structure is shown in Figure 4-1.

Other predicates that are results of actions will have a similar structure, though some may have an additional transition from *true* to *false*. The structure could be

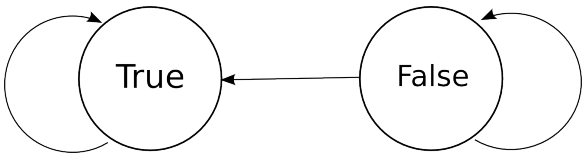


Figure 4-1: Structure of the PCA for the isclean predicate

inferred from the available actions, however, the models are currently constructed by hand. Note that one of these PCAs must be constructed per location that can be cleaned for a particular problem. For the specific ECA problem used here, there are four isclean PCAs, one for each target.

Additional work must be done for the predicates that can be monitored based on sensing information. These predicates are grouped into sets where only one proposition can be true at a time. For example, a manipulator can either be empty or holding a single object. Those that cannot be grouped into mutually exclusive sets of this form are built using a PCA with *true* and *false* modes, similar to isclean above. The grouping is currently performed by hand, though invariant synthesis could also be used to discover these sets [6]. This grouping is performed only for the predicates monitored based on sensed data because the other predicate models are handled as described above. The guard conditions here are built in terms of discrete primitives rather than action statuses and the transitions will be probabilistic rather than deterministic as above.

Those that can be grouped into mutually exclusive sets are more complicated. They obviously have more states and transitions than the standard *true*/*false* model and the transition themselves tend to be more complicated. The structure of the transition function depends is quite domain specific. We will examine the (empty ?manip - manipulator) and (holding ?obj - object ?manip - manipulator) set as an example of the structure.

Intuitively, for a manipulator, in this case a hand, to be holding something, it must overlap that object and move with it. This translates to partially occluding or proper part and stable when written in terms of RCC and QTC. The orientation and relative position of the object and the hand does not matter as long as these two relations

hold, so the rotational primitives do not apply, nor do the other Cartesian relations. Thus formally, `(holding ?obj - object hand)` maps to $((\texttt{PO hand ?obj}) \vee (\texttt{iPP hand ?obj})) \wedge (\texttt{ST hand ?obj})$

In constructing the overall structure of the PCA, we exploit knowledge of how spatial relations evolve over time. For example, the hand cannot go from holding an object to holding another object without putting the first object down. Thus we set up the structure such that from one of the holding states, the only two transitions available are to empty or back to the same holding state. From empty, it is possible to transition to any of the holding states. A transition to a particular holding state is more likely when the specific conditions are met: $((\texttt{PO hand ?obj}) \vee (\texttt{iPP hand ?obj})) \wedge (\texttt{ST hand ?obj})$. If multiple conditions are met, a transition to either of the relevant states is equally likely. In all cases, we keep the probability of a self transition reasonably high to help combat noisy sensors. This structure is shown in Figure 4-2.

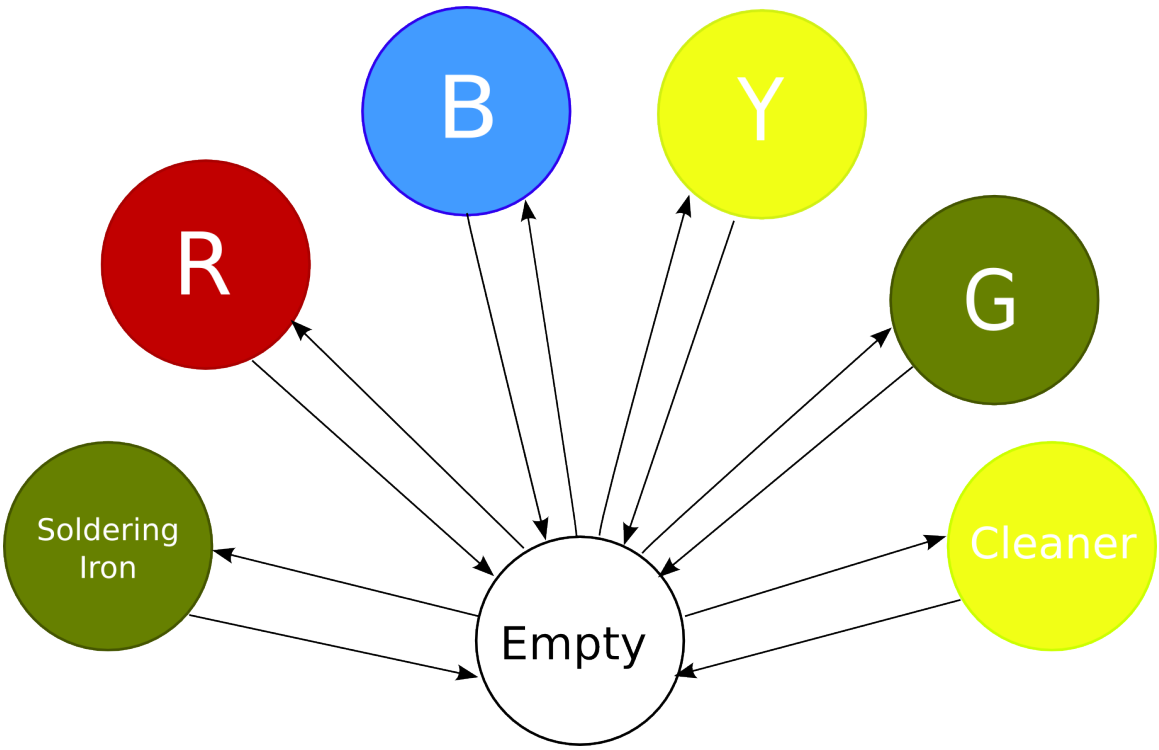


Figure 4-2: Example of a group of mutually exclusive predicates, the hand can only hold one of the components at a time

4.3.2 Durative Actions

Each durative action has the same basic structure with five modes: *ready*, *executing*, *finished*, *failed*, and *stopped*. There is one tPCA model for each grounded action. The structure of the action models is shown in Figure 4-3. In addition to these modes, each action has a clock variable and a single control variable, stop, with a domain of {True, False}. The stop variable is the same for each action and is used to stop and start monitoring.

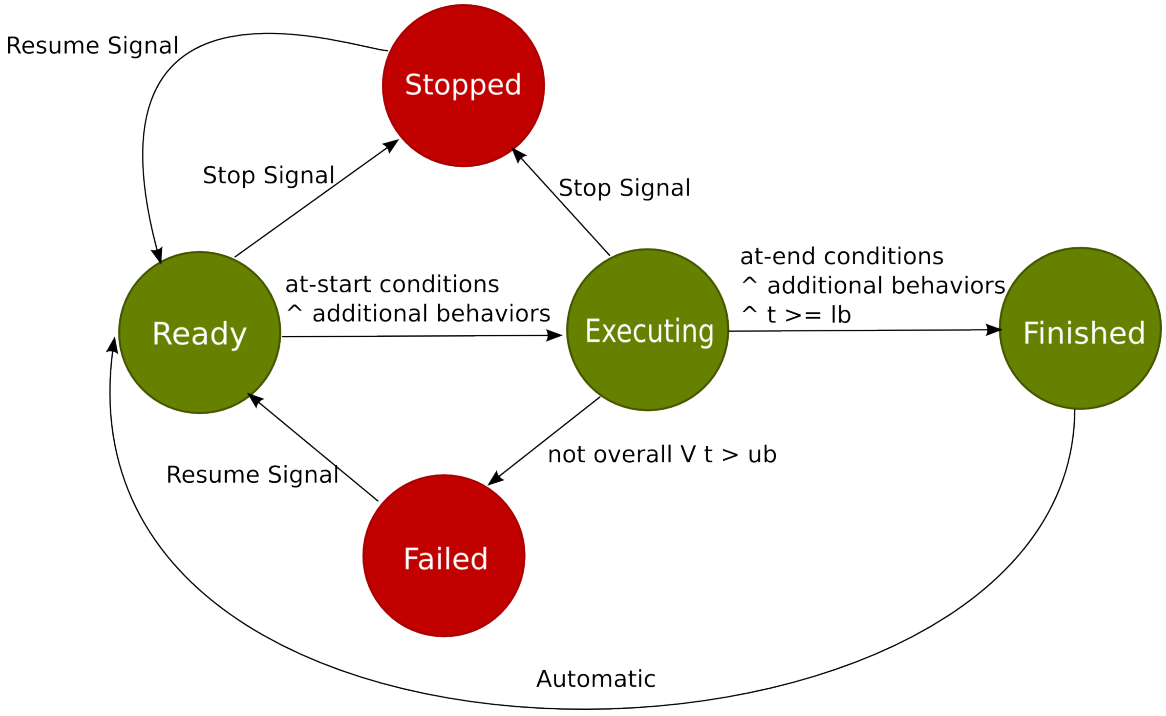


Figure 4-3: Generic Action tPCA structure, pictured without self transitions

In order to determine when an action has started or ended, we at the state of the conditions and certain additional behaviors. The additional behaviors are separate from the conditions and are not defined in the PDDL domain. Instead, these behaviors signal a particular action starting or ending. For instance, before cleaning a particular location, the hand and cleaner will move towards that area and eventually be in the vicinity of that area. When the cleaning has finished, the hand and the cleaner will move out of that area and towards the next location.

The *ready* mode is the default mode while monitoring is in progress. From it, tran-

sitions are possible to the *executing* and *stopped* modes. The transition to *executing* occurs when the at start conditions are met and the starting additional behaviors are observed. The transition to *stopped* occurs when the stop control variable is set to True.

The *executing* mode is used to indicate that an action is in progress. From it, transitions are possible to the *failed*, *finished* and *stopped* modes.

The finished mode is simply used for the application of at end effects. From this mode, a transition to the *ready* mode will occur with 100% probability if the stop variable is not true and to the *stopped* mode if a stop signal is received.

Standard PDDL problems do not consider actions able to fail but in a real world application where actions are not directly controllable, as is the case with systems that involve humans, failures can happen. In order to be able to monitor for this, we created the *failed* mode. The system can only enter *failed* from the *executing* mode. We define failure here as violating the overall constraints, or violating the time duration. If the action is not observed to have finished and the clock variable exceeds the upper bound of the duration, the action is assumed to have failed. We thus define the guard condition for transitioning into this mode as the current time being greater than the upper bound and the conditions marked as over all not holding.

The application of effects is handled by the proposition models and is therefore discussed in Section 4.3.1.

4.4 Chapter Summary

In this chapter, we discussed the process of monitoring PDDL propositions and actions. We define a PCA or tPCA for each proposition and action that could occur for a given problem, and then perform belief state update on the overall set of models, in order to infer the current status of the predicates and actions. The propositions are grouped into mutually exclusive sets where only one can hold at a time. One mode of the PCA is assigned to each possible proposition being true. The transitions between the modes depend on the state of the discrete primitives.

The actions on the other hand have the same transition structure. The only difference between the actions are the transitions thta are defined as a combination of their conditions and additional signaling behaviors. The modes in the action tPCA are: *ready*, *executing*, *finished*, *failed*, and *stopped*. This structure captures the nominal flow of an action as well as two anomalous modes. The *failed* mode captures a limited but still useful form of failure. It is entered when the overall condition is violated or when the upper time bound is violated.

In the next chapter, we will discuss a machine learning technique that allows us to acquire PHA models for discrete primitives from training data. This reduces the amount of work required to implement new discrete primitives.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 5

Model Learning

The structure of the PHCA and tPCCA models discussed thus far can be generated from the PDDL domain descriptions. The transition probabilities and guard conditions however must be acquired either through machine learning or manual estimation. To reduce the amount of knowledge engineering required, we provide a machine learning approach. Transition probabilities and guard conditions need to be learned for both the PHCA models and tPCCA models. In this chapter we examine PHCA learning rather than tPCCA; however, the approach to learning the guard conditions and transition probabilities that is discussed here could be extended to tPCCA models.

In order to reduce the work required to implement new models, we apply techniques from machine learning to learn the unknown parts of a PHA model. In section 5.1 we discuss learning a specific form of PHA model, which is sufficient for modeling the discrete primitives used in this thesis, in an unsupervised manner. Finally, in section 5.2, we discuss learning specific PHA subsets using various levels of supervision.

5.1 Unsupervised Learning of PHA Models

Automated learning of PHCA models is a complicated process which grows more complex as the number of components increases. Instead of solving the problem of learning a large set of interconnecting models, we simplify the problem by isolating

each individual component and focusing on learning each individual PHA separately. This works well for learning models of discrete relations and QSRs as the overall model for a given scenario will tend to have the same structure and similar parameters repeated multiple times to account for all the necessary pairwise relations.

Several others, including Gil and Blackmore [17] [8], have addressed learning PHA models automatically from data, but they did not address the problem of learning PHA models with guarded transitions. We derived an algorithm for learning a simplified form of PHA models that includes learning guard conditions [34].

We focused on learning models of the form:

$$x_i = A_{m_{i-1}} x_{i-1} + B_{m_{i-1}} u_{i-1} + w_{i-1}, \tag{5.1}$$

$$y_i = C_{m_i} x_i + D_{m_i} u_i + v_i, \tag{5.2}$$

$$p(m_{0:n} | x_{0:n}, u_{1:n}) = p(m_0) \prod_{i=1}^{n} p(m_i | m_{i-1}, g_{m_{i-1}}(x_{i-1}, u_{i-1})) \tag{5.3}$$

And further simplify the problem by assuming that the matrices in the observation function C_{m_i} and D_{m_i} are already known and that we have data for the continuous state and exogenous input (x_i and u_i)). Note that A_{m_i} and B_{m_i} model the linear dynamics in discrete mode m_i, $w_{i-1} \sim N(0, Q_{m_{i-1}})$, $v_i \sim N(0, R_{m_i})$ are uncorrelated, white Gaussian noise, and $g_{m_i}(x_i, u_i) \in \mathbb{M}$ is the guard function for mode m_i. This form is sufficient to model the discrete primitives used in this thesis but the use of linear dynamics vastly simplifies several of the steps of the learning process.

We frame the learning problem as an optimization problem with the goal being to find the optimal set of PHA parameters θ^* such that

$$\theta^* = \arg\max_{\theta'} f(\theta') = \log p(y_{1:n} | u_{1:n}; \theta') \tag{5.4}$$

It is not possible to directly optimize this function. Instead, we use Expectation Maximization (EM) to maximize the lower bound:

$$h(\theta')=\sum_{m_{0:n}}\int_{x_{0:n}} p(x_{0:n}, m_{0:n}|y_{1:n}, u_{1:n})L(\theta')dx_{0:n} \tag{5.5}$$

$$=E_{p(x_{0:n},m_{0:n}|y_{1:n},u_{1:n})}[L(\theta')]. \tag{5.6}$$

$$\begin{aligned} L(\theta') &= \log p(y_{1:n}, x_{0:n}, m_{0:n}|u_{1:n}; \theta') \\ &= \log p(x_0, m_0) + \sum_{i=1}^{n} \log p(y_i|x_i, m_i, u_i; \theta') \\ &+ \log p(x_i|x_{i-1}, m_{i-1}, u_{i-1}; \theta') \\ &+ \log p(m_i|m_{i-1}, g_{m_{i-1}}(x_{i-1}, u_{i-1}); \theta') \end{aligned} \tag{5.7}$$

EM splits the optimization into two steps which it repeats until convergence. This method is guaranteed to converge to a local optimum. The two steps are:

1. Use parameters θ^k from the k-th iteration to compute posterior probabilities $p^{k+1}(x_{0:n}, m_{0:n}|y_{1:n}, u_{1:n})$. This is the *E-step*;

2. Use $p^{k+1}(x_{0:n}, m_{0:n}|y_{1:n}, u_{1:n})$ to find new parameters θ^{k+1} that maximize (5.6). This is the *M-step*. Repeat until successive evaluations of (5.6) converge.

Because we are assuming that the continuous state vector can be measured directly, the only hidden state is the mode sequence. Because of this our final objective function is:

$$Q(\theta')=\sum_{m_{0:n}} p(m_{0:n}|x_{0:n}, u_{1:n})\tilde{L}(\theta') \tag{5.8}$$

$$=E_{p(m_{0:n}|x_{0:n},u_{1:n})}[\tilde{L}(\theta')]. \tag{5.9}$$

5.1.1 E Step

The log-likelihood of the data in 5.9 is:

$$\begin{aligned}\tilde{L}(\theta) &= \log p(x_{0:n}, m_{0:n}|u_{1:n};\theta) \\ &= \log p(x_0, m_0) + \sum_{i=1}^{n} \log p(x_i|x_{i-1}, m_{i-1}, u_{i-1};\theta) \\ &+ \sum_{i=1}^{n} \log p(m_i|m_{i-1}, g_{m_{i-1}}(x_{i-1}, u_{i-1});\theta) \end{aligned} \tag{5.10}$$

In order to estimate our objective function, we first define posterior marginal mode probabilities, $\gamma(m_{i-1})$ and $\xi(m_{i-1}, m_i)$ as:

$$\begin{aligned}\gamma(m_{i-1}) &= p(m_{i-1}|x_{0:n}, u_{1:n};\theta), \\ \xi(m_{i-1}, m_i) &= p(m_{i-1}, m_i|x_{0:n}, u_{1:n};\theta), \end{aligned} \tag{5.11}$$

We can use a forward-backward algorithm to compute these values. We define the forward value, $\alpha_g(m_{i-1})$, and the backward value, $\beta_g(m_{i-1})$, as

$$\begin{aligned}\alpha_g(m_{i-1}) &= p(x_1, \ldots, x_{i-1}, m_{i-1}|u_{1:n};\theta), \\ \beta_g(m_{i-1}) &= p(x_n, \ldots, x_i|m_{i-1}, x_{i-1}, u_{1:n};\theta). \end{aligned} \tag{5.12}$$

We also define a dynamics probability $d(x_i, m_{i-1})$, which is computed from the dynamics, and a transition probability, which is the probability of transitioning from one mode to another. Formally these are:

$$\begin{aligned}d(x_i, m_{i-1}) &= p(x_i|x_{i-1}, m_{i-1}, u_{i-1}), \\ t(m_{i-1}, m_i) &= p(m_i|m_{i-1}, g_{m_{i-1}}(x_{i-1}, u_{i-1})). \end{aligned} \tag{5.13}$$

Alpha and beta can then be computed recursively as:

$$\alpha_g(m_{i-1}) = \sum_{m_{i-2}} \alpha_g(m_{i-2}) d(x_{i-1}, m_{i-2}) t(m_{i-2}, m_{i-1}), \tag{5.14}$$

$$\beta_g(m_{i-1}) = \sum_{m_i} \beta_g(m_i) d(x_i, m_{i-1}) t(m_{i-1}, m_i), \tag{5.15}$$

$$\alpha_g(m_0) = p(x_0, m_0), \beta_g(m_n) = 1. \text{ (given)}.$$

We can then write $\gamma(m_{i-1})$ and $\xi(m_{i-1}, m_i)$ as:

$$\gamma(m_{i-1}) = \frac{\alpha_g(m_{i-1})\beta_g(m_{i-1})}{\sum_{m'_{i-1}} \gamma(m'_{i-1})}, \tag{5.16}$$

$$\xi(m_{i-1}, m_i) = \frac{\alpha_g(m_{i-1}) d(x_i, m_{i-1}) t(m_{i-1}, m_i) \beta_g(m_i)}{\sum_{m'_{i-1}, m'_i} \xi(m'_{i-1}, m'_i)},$$

Our final objective function is then:

$$Q(\theta) = E_{p(m_{0:n}|x_{0:n}, u_{1:n})}[\tilde{L}(\theta)]$$

$$= \sum_{i=1}^{n} \left(\sum_{m_{i-1}} \gamma(m_{i-1}) \log p(x_i | x_{i-1}, m_{i-1}, u_{i-1}) + \sum_{m_{i-1}, m_i} \xi(m_{i-1}, m_i) \log p(m_i | m_{i-1}, g_{m_{i-1}}(x_{i-1}, u_{i-1})) \right)$$

$$+ \sum_{m_0} \gamma(m_0) \log p(x_0, m_0), \tag{5.17}$$

5.1.2 M Step

In the M step, we then need to optimize the objective function, which was computed in the E step. In this section we show the maximization process for each parameter.

Initial Mode Probability

This is simply:

$$\hat{p}(m_0 = k) = \frac{\gamma(m_0 = k)}{\sum_{k'} \gamma(m_0 = k')} \tag{5.18}$$

Transition Probabilities

The guarded transition probabilities are given by:

$$\hat{p}(m_i{=}k | m_{i-1}{=}k', g_{m_{i-1}=k}(x_{i-1}, u_{i-1}){=}m){=} \frac{\sum_{\{x_{j-1}, u_{j-1}\}: g_{m_{j-1}=k}(x_{j-1}, u_{j-1})=m} \xi(m_{j-1}{=}k', m_j{=}k)}{\sum_{m_j=k''} \hat{p}(m_j{=}k'' | m_{j-1}{=}k', g_{m_{j-1}=k}(x_{j-1}, u_{j-1}){=}m)} \tag{5.19}$$

Guard Functions

Learning the guard functions involves creating a function that maximizes the following:

$$m^*{=}\arg\max_m \sum_{m_i} \xi(m_{i-1}, m_i) \log p(m_i | m_{i-1}, g_{m_{i-1}}{=}m). \tag{5.20}$$

In order to do so, we first compute the mapping $(x_{i-1}, u_{i-1}) \rightarrow m \in \mathbb{M}$ by means of 5.20 and treat this as "labeled" data to use in training a multiclass classifier.

In this work we chose to use a multiclass support vector machine (SVM) as our classifier. SVMs have many advantages but in particular they are useful because they can represent an arbitrarily shaped transition functions. Additionally, they can be optimized quickly for moderately sized data-sets. Further discussion of training SVMs is out of the scope of this work but is widely available and extensive literature is available on the subject. Useful resources include Burges [11] and Bishop. [7].

Linear Models

From the dynamics equations, we know that:

$$p(x_i|x_{i-1},m_{i-1},u_{i-1};\theta) = N(x_i; A(m_{i-1})x_{i-1} + B(m_{i-1})u_{i-1}, Q) \tag{5.21}$$

And because Q is fixed and uncorrelated, we have:

$$\log p(x_i|x_{i-1},m_{i-1},u_{i-1};\theta) = -\frac{1}{2\sigma^2}\|x_i - \mu_{i-1}\|^2 + c, \tag{5.22}$$

Where $\mu_{i-1} = A(m_{i-1})x_{i-1} + B(m_{i-1})u_{i-1}$. The parameters that we are trying to learn are A_k and B_k. Selecting their optimal value requires us to minimize:

$$(Y - \Phi b(k))^T W(k)(Y - \Phi b(k)), \tag{5.23}$$

$$\Phi = \begin{bmatrix} x_0^T & u_0^T \\ \vdots & \vdots \\ x_{n-1}^T & u_{n-1}^T \end{bmatrix}, Y = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}, b(k) = \begin{bmatrix} A_k^T \\ B_k^T \end{bmatrix}$$
$$W(k) = diag(\gamma(m_0){=}k, \cdots, \gamma(m_{n-1}){=}k), \tag{5.24}$$

Which is simply Weighted Least Squares (WLS). The closed form solution is then:

$$\hat{b}(k) = \begin{bmatrix} \hat{A}_k^T \\ \hat{B}_k^T \end{bmatrix} = (\Phi^T W(k)\Phi)^{-1}\Phi^T W(k)Y. \tag{5.25}$$

5.2 Supervised Learning of PHA Models

In the previous section, we assumed that we had no knowledge of the parameters or the forms of the equations but in practice, this is not always true. In most domains, there is some structure that is assumed or some additional data that is collected. Using the unsupervised approach described above as a starting point, we now examine how the learning process is affected when certain parameters are known.

5.2.1 Dynamics Known

If the dynamics are known, the steps described in section 5.1.2 can be skipped. We still require that the continuous state x_i be observable but the form of the dynamics is much more flexible. Assuming we do not wish to learn the parameters, we can use the more general form:

$$x_i = f_m(x_{i-1}, u_{i-1}) + v_i \tag{5.26}$$

With the noise term v_i still assumed to be uncorrelated white Gaussian noise, the calculation of α, β, γ, and ξ and the remaining optimizations in the M step remain unchanged.

5.2.2 Guard Conditions and Transition Probabilities Known

If the Guard Conditions and Transition Probabilities are know, the only change is that the steps described in sections 5.1.2 and 5.1.2 can be skipped.

5.2.3 Mode Labels Known

If the mode labels are known, there are no hidden states and the learning can be done simply by running one iteration of the EM algorithm. The values of α, β, γ, and ξ will not be updated as the mode sequence in the training data is known with some

accuracy. Simply computing each of the parameters as described in the M step will arrive at the optimal values given the information that is known.

5.3 Chapter Summary

In this chapter we derived an algorithm for learning PHA models from data. The approach uses expectation maximization to incrementally improve the fit of the model until it converges at a local optima. We also discussed several modifications to the basic EM algorithm that allow for a partially supervised approach. Together, this allows us to learn PHA models for the discrete primitives with limited overhead, reducing the engineering that goes into getting LCARS running on a new domain. In the next chapter, we will validate the overall LCARS approach on the ECA scenario, discussing the implementation, experimentation and results.

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 6

Experimental Design and Results

In this chapter, we validate the approach presented in this thesis, discussing the implementation, experiments and results. In section 6.1, we discuss the implementation of LCARS on the ECA scenario, describing the testbed and the implementations of each of the filter components. In section 6.2 we discuss a set of experiments to validate the learning algorithm introduced in Chapter 5. In section 6.3 we discuss a set of experiments that validate the activity recognition and propositional monitoring approaches that are central to this thesis. In sections 6.4 and 6.4.3 we present and summarize the results of the experiments.

6.1 Implementation

6.1.1 Testbed

We designed a testbed on which the ECA scenario could be implemented. The robot used is a Rethink Robotics Baxter which has two arms with 7 degrees of freedom. The tools and objects were represented by foam blocks due to the limitations on what the Baxter end effectors can reasonably pick up. Communication with the robot was done using several libraries within the Robot Operating System (ROS). We put fiducial tags [27] on each of the objects in order to track them and constructed a glove with a fiducial tag on it in order to track the position of the hand. Web-cams were

mounted on a frame over the testbed area in order to track the tags. The position and orientation of the tags were estimated using the ROS wrapper for Alvar, an open source library for fiducial tag tracking. The testbed configuration is shown in Figure 6-1.

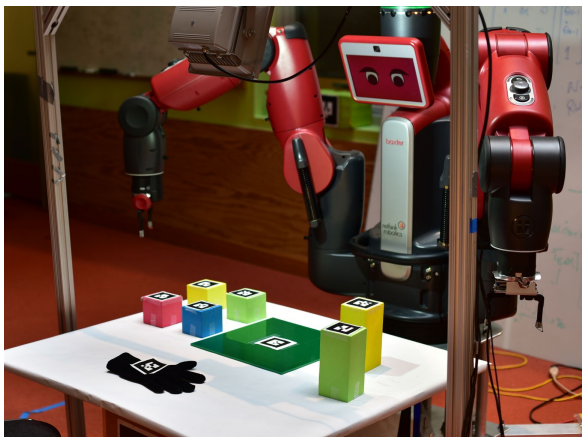


Figure 6-1: Robotic Testbed

When all of the discrete primitives, predicates and actions are modeled for the ECA scenario, there are a total of: 100 discrete primitives, 46 propositions and 80 possible actions.

6.2 PHA Learning Experiments

We implemented the learning algorithms using Python and the multi-class SVM's were trained using *Scikit-learn* [30]. Several test systems were used including several examples relevant to activity recognition and several pedagogical examples. In this section we discuss each of the test systems and discuss how data was collected for each system.

6.2.1 Test Systems

The test systems described here were selected to show a variety of systems on which the model learning algorithm can be applied. The first test systems directly relate to LCARS with models for RCC, QTC and the handoff task being learned. The other

two examples are pedagogical. The first is an RC circuit which was selected for its simplicity and the second is a target tracking example which was selected because of its use in the hybrid model literature.

QSRs

The third test system are the RCC and QTC relations for several of the objects used in the ECA scenario. We set up the visual tracking system as described in Section 6.1.1. We collected training data by moving objects around the environment as we would during an actual ECA run. We recorded the video as well as the outputs of the sensing system and then labeled the data after the fact by watching the videos and noting the various QSRs. We then partitioned the data in the various runs into cases where there was at least one transition and grouped together the data for each type of relation (RCC with a particular object class pairing or QTC). We ran our learning algorithm on the collected data using the unsupervised case and the case where the mode labels are known.

Handoff

Our final test system is the handoff task as described by the Pass action within the ECA scenario. In this scenario, the Baxter arm begins holding an object at a fixed position the human then reaches out and grabs the objects. The discrete state is whether or not the human is holding the object and the continuous state is the torque in the wrist joint of the Baxter's arm. We collected data by repeatedly grabbing and releasing the object while the Baxter sat at a fixed position. We used a button on the Baxter to indicate moments where the object was grasped, pressing it when we were holding the object and releasing it when we weren't. We did this for several different positions though the model learned is specific to one particular joint configuration. We ran our learning algorithm on the collected data both in the unsupervised case and the supervised case where the mode data was labeled. This is intended to illustrate that it is possible to learn new domain specific discrete relations.

Switched Resistor-Capacitor Circuit

The first test system is a simple resistor capacitor circuit, the layout of which is shown in Figure 6-2. This is a clean and simple example of a hybrid system. The continuous mode is the output voltage and the discrete mode is the switch status. The input to the system is the input voltage.

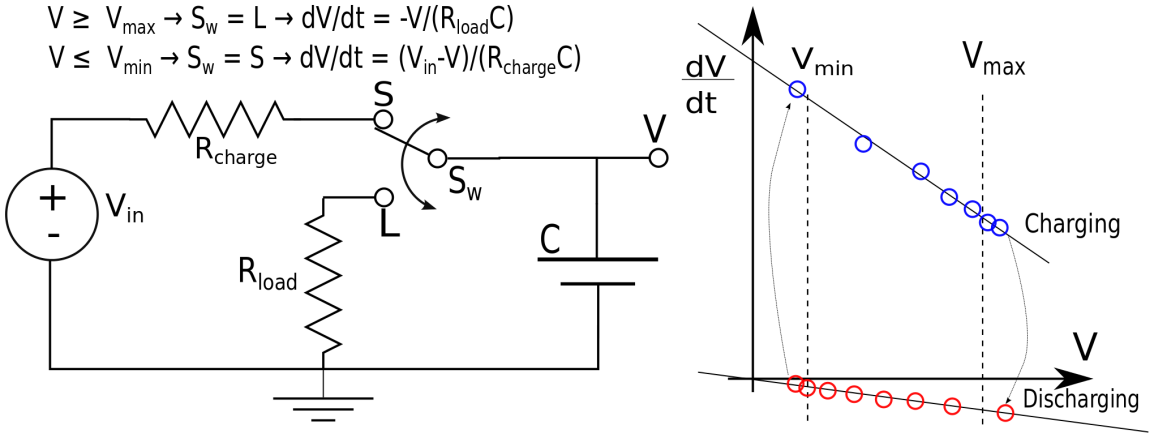


Figure 6-2: Switched RC circuit diagram.

We generated training data for this system by simulating several runs with the minimum and maximum input voltages set to respectively and the input voltage set to . We ran our learning algorithm on this generated data completely unsupervised and with the mode labels known as described in section 5.2.

Airplane

The second test system is a target tracking problem, a classic benchmark problem for hybrid filtering. Our models were based on the models of Seah and Hwang [36] which describe an airplane that can travel forward, turn left and turn right, all at constant rates. We generated data from these models using two types of flight paths, a "lawnmower" pattern and a random path. In the lawnmower pattern, the vehicle travels back and forth over a bounding box. This type of pattern is used in the real world for crop dusting and some kinds of exploration [18, 10, 15]. An example of this pattern is shown in Figure 6-3. This pattern is meant to illustrate how the learning algorithm will learn patterns in the transitions such as the edges of the bounding box.

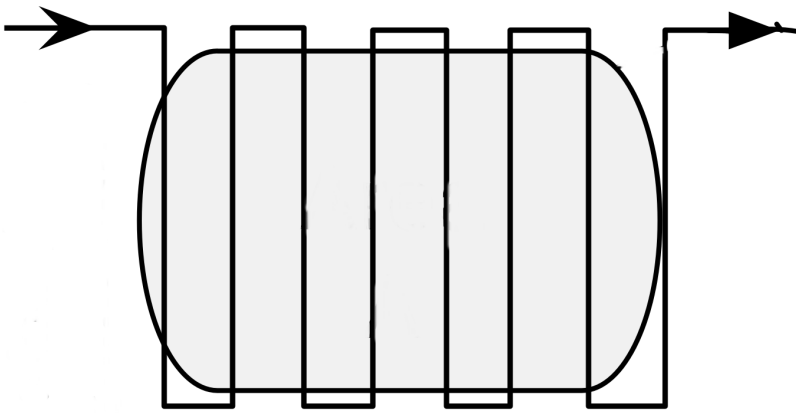


Figure 6-3: Lawnmower Pattern

The random flight path was designed to show a more complex path that does not follow any sort of pattern. In it the vehicle chooses one of the three actions randomly with equal probability, performs that action for 50 time steps, and then independently chooses another action and performs that for 50 time steps. This is meant to illustrate transitions that do not happen with any pattern that would be discernible by our learning process.

For each of these patterns we generated several test runs. We then ran our learning algorithm on the generated data unsupervised and with the mode labels known as described in section 5.2.

6.2.2 Performance Metrics

We used the implemented IMM filter to assess each of the test systems.We calculated the following test statistics in each case:

- Percentage of misclassified modes
- Mean delay from mode change to estimated mode change

These were selected in order to showcase the accuracy and speed of the learned PHA models. We focus on the discrete mode rather than the continuous state because the mode is the more important factor for LCARS.

6.3 Action and Predicate Recognition Experiments

In order to assess the action and predicate recognition capabilities, we implemented the ECA scenario on the testbed described above. Since we wanted to evaluate how well the system performs on human actions, we removed the *pass* action. The *pass* action requires both robot and human action and is instantiated by the robot. We can only monitor the end of the action and this is dependent on the *gripped*/*not gripped* predicate. We describe how we tested the recognition of that predicate above. The other four actions, *pick*, *place*, *solder* and *clean* were still used.

We generated several possible executions of the scenario using the Temporal Fast Downward planner [20] and recorded them being performed. The position data was recorded as well as the raw video. The video was used to determine and label the ground truth for the QSRs, predicates and actions in each case.

6.3.1 Performance Metrics

We implemented the filters described above in order to assess performance. We calculated the following test statistics:

- Number of correctly and incorrectly classified actions
- Mean delay from action start to estimated action start
- Mean delay from action end to estimated action end

These test statistics were selected to show the speed and accuracy of LCARS. The status of the actions was selected over the status of the predicates because it was easier to establish the ground truth.

6.4 Results

Note: All values are rounded to the nearest two decimal places.

6.4.1 PHA Learning Experiments

Table 6.1: Unsupervised Learning Results

	Misclassified Modes (%)	Mean Mode Change Delay (# of Time Steps)
RC Circuit	1.23	9.63
Airplane	3.33	3.20
RCC	53.1	n/a
QTC	46.3	n/a
Handoff	64.2	n/a

Table 6.2: Supervised Learning Results, Mode Labels Known

	Misclassified Modes (%)	Mean Mode Change Delay (# of Time Steps)
RC Circuit	1.25	8.43
Airplane	1.15	2.32
RCC	9.34	7.51
QTC	8.32	6.44
Handoff	3.23	6.96

6.4.2 Action and Predicate Recognition Experiments

Table 6.3: Combined Results for all actions

Correctly Classified Actions (%)	84.8
Mean Action Start Delay (s)	0.54
Mean Action End Delay (s)	1.75

Table 6.4: Results for the pick actions

Correctly Classified Actions (%)	83.33
Mean Action Start Delay (s)	0.47
Mean Action End Delay (s)	1.57

Table 6.5: Results for the place action

Correctly Classified Actions (%)	84.61
Mean Action Start Delay (s)	0.43
Mean Action End Delay (s)	1.64

Table 6.6: Results for the clean action

Correctly Classified Actions (%)	75
Mean Action Start Delay (s)	0.61
Mean Action End Delay (s)	1.92

Table 6.7: Results for the solder action

Correctly Classified Actions (%)	75
Mean Action Start Delay (s)	0.67
Mean Action End Delay (s)	1.86

6.4.3 Summary of Results

Tables 6.1 and 6.2 show the results for the learning experiments. The performance on the monitoring related areas was poor for the unsupervised task. We believe that this is because it was over-fitting to local optima where the learned mode had nothing to do with the mode we wanted to learn. In the handoff task it particular, it was observed that the model that was learned in the unsupervised mode classified an increase in torque as mode 0 and a decrease in torque as mode 1. This is completely unrelated to the gripped and not gripped modes we wanted it to learn. The mode change delay for those tasks is listed as n/a because the performance was poor enough that the test statistic could not be reliably calculated.

The performance of the learning task with the modes supplied supports this hypothesis as the learned models showed a reasonable level of performance for all five test systems. This shows that we are able to learn models that are useful for monitoring discrete primitives from data if a reference mode set is supplied.

Regarding the action and predicate monitoring, a direct comparison to other state of the art monitors would be challenging to make given that each was implemented on a different set of actions. Without implementing the other approaches on the ECA scenario or implementing LCARS on other domains, we cannot directly compare the results. That being said, we can make observations about the performance of LCARS on the ECA scenario and examine the relative accuracy of approaches on their own domains.

LCARS was often able to correctly classify the action that was being performed

and it was able to do so with only a short delay. When it incorrectly classified an action, it was generally confusing one action with another action of the same type. For instance, it would sometime confuse picking up the red block and picking up the blue block. In a large number of the incorrectly identified cases, the predicates were soon correctly identified after the action was finished such that an inconsistent state was soon detected by Pike. In these cases, the failure in action monitoring was recoverable. This was not the case for the solder and clean actions that were incorrectly identified.

With regards to the accuracy of the action monitoring, table 6.8 shows the classification accuracy of a number of different action recognition approaches including LCARS. It is not useful to directly compare the percentages as the domains are wildly different but we can make some overall observations as to the relative performance of LCARS.

Table 6.8: Accuracy of Various Activity Recognition Approaches

	Correctly Classified Actions (%)
LCARS	84.8
Ravi [32]	90.61
Patel [29]	82.13
Padoy [28]	93.5 - 99.6
Behera [3]	61.10

The Ravi paper used a machine learning classification scheme to recognize a set of human actions in daily life from accelerometer data. They were able to distinguish between a set of 8 possible actions with 90.61% accuracy.

The Patel paper uses a machine learning classification scheme to classify time series data as belonging to particular actions. They used two domains, the first being recognizing gestures in American Sign Language and the second being recognizing stages of Hepatitis treatment. Over both of these domains they showed 82.13% accuracy rate which is comparable in magnitude to LCARS. The ASL domain that they used had a much larger number of possible actions than LCARS and the Hepatitis dataset had a much smaller number of actions.

The Padoy paper used hierarchical HMMs to monitor actions in an operating

room. They split their overall accuracy results between different phases of their workflow and did not give an overall result. In their worst phase they had 93.5% accuracy and in their best phase they had 99.6% accuracy. This certainly outshines LCARS and shows a much higher level of reliability.

Of the approaches listed in this table, the Behera approach is the closest to LCARS as it uses QSRs to perform workflow monitoring. In addition, their domain was a human manufacturing task. Their relatively low percentage of correct recognition is due to the way they calculated their test statistic. The other papers and LCARS all had a binary for any given run, either the action was correctly classified or it wasn't. The Behera paper on the other hand looked at every moment during each of their runs and determined if they had correctly classified the action. This resulted in a much lower accuracy than many of the other papers.

Overall, LCARS performed reasonable well on the ECA domain but there is still room for improvement. The current performance serves as a proof of concept that this strategy can be applied to a real system but does not necessarily show any significant improvement over the state of the art.

Chapter 7

Results and Conclusions

7.1 Summary of Contributions

LCARS approaches the problem of predicate and action estimation. It splits this into two key components: a discrete primitive monitor and a predicate and action monitor. The first piece converts the raw sensor data into discrete primitives, simple, semantically significant relationships between objects. The set of primitives for a particular problem is mapped into a cPHA upon which belief state update is performed. The cPHA representation of the QSRs was introduced in this work and was shown to be effective for recognizing RCC and QTC relationships.

The predicate and activity monitor uses the output of the discrete primitive filter as input. It maps the status of the discrete primitives to the guard conditions of the predicates and maps both the primitives and the predicates to the actions. A PCA or tPCA is constructed for each predicate group and each possible action for a particular PDDL problem and belief state update is performed over the entire tPCCA. This belief state is given to an external executive. The tPCCA used for the actions and predicates in this work builds on previous uses of tPCCAs to represent PDDL problems by adding mechanisms for performing recognition [41]. LCARS also differs from the state of the art in that it reasons about the state of the world and the state of the actions using a representation that is also commonly used by planners.

We also derived and demonstrated an algorithm for learning PHA models from

data. We expanded on an unsupervised EM based algorithm by adding several modifications to facilitate a semi-supervised learning approach. This allowed the algorithm to be applied to the domains required for discrete primitive monitoring.

7.2 Future Work

Though LCARS performed well on this problem, there are still many ways it can be improved. First, rather than using an IMM filter, a modern approach to performing cPHA estimation such as A* with Bounding Conflicts [39] should be used. Additionally, the performance of this approach should be evaluated using the full tPCCA model rather than the HMM decomposition used for the experiments. This should improve the accuracy.

Additionally, the tPCCA models take a lot of time to create by hand. Most of the pieces of these models can likely be learned through demonstration data. The guard conditions can likely be learned through a process similar to that described in Chapter 5. The predicates for a particular problem can likely be grouped into mutually exclusive sets using invariant synthesis [6]. Using these structures, a bootstrapping algorithm could be derived that started with the PDDL domain, a set of PDDL problems, and a set of demonstration actions and then learned the predicate models and then learned the activity models. This algorithm would group the predicates using invariant synthesis then learn the guard conditions and constraints to build the predicate models. The predicate models and the existing discrete primitive models would then be used as input to help learn the action models.

Bibliography

[1] James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] Philippe Balbiani, Jean-François Condotta, and Luis Farinas del Cerro. A new tractable subclass of the rectangle algebra. In *IJCAI*, volume 99, pages 442–447. Citeseer, 1999.

[3] Ardhendu Behera, Anthony G Cohn, and David C Hogg. *Workflow activity monitoring using dynamics of pair-wise qualitative spatial relations*. Springer, 2012.

[4] Brandon Bennett. Spatial reasoning with propositional logics. *KR*, 94:51–62, 1994.

[5] Brandon Bennett. Modal logics for qualitative spatial reasoning. *Logic Journal of IGPL*, 4(1):23–45, 1996.

[6] Sara Bernardini and David E Smith. Automatic synthesis of temporal invariants. In *SARA*, 2011.

[7] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[8] Lars Blackmore, Stephanie Gil, Seung Chung, and Brian Williams. Model learning for switching linear systems with autonomous mode transitions. In *Decision and Control, 2007 46th IEEE Conference on*, pages 4648–4655. IEEE, 2007.

[9] Lars James Christopher Blackmore. *Robust execution for stochastic hybrid systems*. PhD thesis, Massachusetts Institute of Technology, 2007.

[10] Mitch Bryson, Alistair Reid, Fabio Ramos, and Salah Sukkarieh. Airborne vision-based mapping and classification of large farmland environments. *Journal of Field Robotics*, 27(5):632–655, 2010.

[11] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, 2(2):121–167, 1998.

[12] Yann Chevriaux, Eric Saux, and Christophe Claramunt. A landform-based approach for the representation of terrain silhouettes. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, pages 260–266. ACM, 2005.

[13] Anthony G Cohn. Formalising bio-spatial knowledge. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 198–209. ACM, 2001.

[14] Anthony G Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Qualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3):275–316, 1997.

[15] Gabriele Ferri, Michael V Jakuba, and Dana R Yoerger. A novel method for hydrothermal vents prospecting using an autonomous underwater robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1055–1060. IEEE, 2008.

[16] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *J. Artif. Intell. Res.(JAIR)*, 20:61–124, 2003.

[17] S. Gil and B. Williams. Beyond local optimality: An improved approach to hybrid model learning. In *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*, pages 3938–3945, Dec 2009.

[18] Michael A Goodrich, Bryan S Morse, Damon Gerhardt, Joseph L Cooper, Morgan Quigley, Julie A Adams, and Curtis Humphrey. Supporting wilderness search and rescue using a camera-equipped mini uav. *Journal of Field Robotics*, 25(1-2):89–110, 2008.

[19] Roop K Goyal and Max J Egenhofer. Similarity of cardinal directions. In *Advances in Spatial and Temporal Databases*, pages 36–55. Springer, 2001.

[20] Malte Helmert. The fast downward planning system. *J. Artif. Intell. Res.(JAIR)*, 26:191–246, 2006.

[21] Michael W Hofbaur and Brian C Williams. Mode estimation of probabilistic hybrid systems. In *Hybrid Systems: Computation and Control*, pages 253–266. Springer, 2002.

[22] Michel D. Ingham. *Timed Model-based Programming: Executable Specifications for Robust Mission-Critical Sequences.* PhD thesis, Massachusetts Institute of Technology, May 2003.

[23] Benjamin Kuipers and Yung-Tai Byun. A robot exploration and mapping strategy based on a semantic hierarchy of spatial representations. *Robotics and autonomous systems*, 8(1):47–63, 1991.

[24] Steven J. Levine and Brian C. Williams. Concurrent plan recognition and execution for human-robot teams. In *ICAPS-14*, 2014.

[25] Jiming Liu. A method of spatial reasoning based on qualitative trigonometry. *Artificial Intelligence*, 98(1):137–168, 1998.

[26] Ye Liu, Liqiang Nie, Lei Han, Luming Zhang, and David S Rosenblum. Action2activity: Recognizing complex activities from sensor data.

[27] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407. IEEE, 2011.

[28] Nicolas Padoy, Diana Mateus, Daniel Weinland, M-O Berger, and Nassir Navab. Workflow monitoring based on 3d motion features. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 585–592. IEEE, 2009.

[29] Dhaval Patel, Wynne Hsu, and Mong Li Lee. Mining relationships among interval-based events for classification. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 393–404. ACM, 2008.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] Claudio S Pinhanez and Aaron F Bobick. Human action detection using pnf propagation of temporal constraints. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 898–904. IEEE, 1998.

[32] Nishkam Ravi, Nikhil Dandekar, Preetham Mysore, and Michael L Littman. Activity recognition from accelerometer data. In *AAAI*, volume 5, pages 1541–1546, 2005.

[33] Jochen Renz and Debasis Mitra. Qualitative direction calculi with arbitrary granularity. In *PRICAI*, volume 3157, pages 65–74, 2004.

[34] Pedro Santana, Spencer Lane, Eric Timmons, Brian Williams, and Carlos Forster. Learning hybrid models with guarded transitions. In *Association for the Advancement of Artificial Intelligence*, 2015.

[35] Christoph Schlieder. Reasoning about ordering. In *Spatial Information Theory A Theoretical Basis for GIS*, pages 341–349. Springer, 1995.

[36] Chze Eng Seah and Inseok Hwang. State estimation for stochastic linear hybrid systems with continuous-state-dependent transitions: an IMM approach. *Aerospace and Electronic Systems, IEEE Transactions on*, 45(1):376–392, 2009.

[37] Spiros Skiadopoulos and Manolis Koubarakis. On the consistency of cardinal direction constraints. *Artificial Intelligence*, 163(1):91–135, 2005.

[38] Muralikrishna Sridhar, Anthony G. Cohn, and David C. Hogg. Unsupervised learning of event classes from video. In *Proc. AAAI*, pages 1631–1638. AAAI Press, 2010.

[39] Eric Timmons. Fast, approximate state estimation of concurrent probabilistic hybrid automata. Master's thesis, Massachusetts Institute of Technology, June 2013.

[40] Nico Van de Weghe, Anthony G Cohn, Guy De Tre, and Philippe De Maeyer. A qualitative trajectory calculus as a basis for representing moving objects in geographical information systems. *Control and Cybernetics*, 35(1):97, 2006.

[41] David Wang and Brian C. Williams. tburton: A divide and conquer temporal planner. In *Proceedings of the Twenty-Ninth Conference on Artificial Intelligence*, Austin, Texas, January 2015.

[42] Xinxi Wang, David Rosenblum, and Ye Wang. Context-aware mobile music recommendation for daily activities. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 99–108. ACM, 2012.

[43] Brian C Williams, Michel D Ingham, Seung H Chung, and Paul H Elliott. Model-based programming of intelligent embedded systems and robotic space explorers. *Proceedings of the IEEE*, 91(1):212–237, 2003.

[44] Brian C Williams and P Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 971–978, 1996.

[45] Shin-Yi Wu and Yen-Liang Chen. Mining nonambiguous temporal patterns for interval-based events. *Knowledge and Data Engineering, IEEE Transactions on*, 19(6):742–758, 2007.

[46] Yongmian Zhang, Yifan Zhang, Eran Swears, Natalia Larios, Ziheng Wang, and Qiang Ji. Modeling temporal interactions with interval temporal bayesian networks for complex activity recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(10):2468–2483, 2013.