BEHAVIOR MODELING FOR HYBRID ROBOTIC SYSTEMS

by

Chetan Rawal

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

Spring 2011

BEHAVIOR MODELING FOR HYBRID ROBOTIC SYSTEMS

by

Chetan Rawal

Approved: _____

Herbert G. Tanner, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____

Anette M. Karlsson, Ph.D.
Chair of the Department of Mechanical Engineering

Approved: _____

Michael J. Chajes, Ph.D.
Dean of the College of Engineering

Approved: _____

Charles G. Riordan, Ph.D.
Vice Provost for Graduate and Professional Education

ACKNOWLEDGEMENTS

I wish to thank my adviser, Dr. Herbert G. Tanner, for his support during all phases of my research. I owe a lot to him for my concepts in robotics and controls, in addition to technical writing skills. His guidance has been invaluable in shaping my career.

Much success in this interdisciplinary work has been possible due to the contributions of Dr. Jeffrey Heinz in formal language theory. I appreciate his enthusiasm and support that helped me to learn formal languages concepts and apply them to hybrid systems.

I acknowledge the joint work with Dr. Herbert Tanner and Ms. Jie Fu for developing much of chapter 3 of this thesis, that deals with modeling a hybrid robotic system and abstracting it to a discrete (finite) transition system. I also want to thank Mr. Nicholaus Lacock who helped me with control programs in C++ that enabled us to exhibit a demo with a robot fetching a print-out.

My committee members, Dr. Herbert Tanner, Dr. Ioannis Poulakakis and Dr. Jeffrey Heinz, have helped improve my thesis through their useful comments and questions, and I thank them as well. I thank the Mechanical Engineering staff at the University of Delaware, especially Ms. Lisa Katzmire, for their friendly approach and very prompt work. Finally, I thank all my lab-mates, friends and family for their moral support to me.

This work was financially supported by NSF grants number 0907003 and 1035577.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The behavior of a certain class of hybrid robotic systems can be expressed using formal languages. In this work, we show how languages can be generated from discrete abstractions of such hybrid systems; that these languages are regular; and they belong to the star free (SF) class of the Sub-regular hierarchy.

Planning and control of hybrid systems is typically difficult due to the computational cost involved in predicting the system's future states, since the states can take infinite values while evolving along the trajectories of continuous dynamics. A discrete abstraction of the hybrid system can reduce these values to a finite number, thereby fascilitating the solution to the reachability problem. Abstraction enables us to focus on planning the system's overall behavior through controller sequences observed in the abstract system, instead of dealing with the dynamics associated with each controller.

The constraints between controllers enable or disable their temporal sequencing. Similarity of these constraints with those found in formal language theory, allows us to express controller sequences as strings of symbols forming a formal language. A formal language analysis of hybrid systems provides an approach for automatic planning and control design synthesis in single and multi-agent robotic systems.

The class of hybrid systems considered in this work have convergent continuous dynamics with parameterized attractors. We model a robot as a hybrid system, and abstract the hybrid system to a discrete transition system. Plans of controller sequences generated on the transition system are implementable on the

hybrid system because of a (weak) bisimulation established between the two systems. Constraints are identified between controllers, that affect their sequencing, with each constraint forming a sub-regular class of controller sequences. Intersection of these languages yield *(sub)regular robotic languages* that express the overall behavior of the underlying hybrid system.

Other models of robot (motion) control such as motion description languages and linear temporal logics generate regular and $\omega-$regular languages respectively. Subregular languages, generated by our classes of hybrid systems, offer structure that can be exploited to operate on system representations in a way that reigns in the complexity of the outcome. The technical contribution of this work in the field of analysis of hybrid systems is that it identifies classes of hybrid robotic systems that can be abstracted so that their overall behavior can be described using subregular languages, and characterizes these languages within the Chomsky hierarchy.

This work contributes also to the formal language community by defining a new class of subregular languages, called the *tier-based strictly local languages*, which captures long-distance constraints between symbols. The tier-based language models have existed in phonology, especially in the form of *autosegmental patterns*. However, these models have primarily dealt with expressing certain phonological patterns on tiers, instead of analyzing the tiers, as our work does here.

This work opens ventures for exploring learning of the regular robotic languages by using phonological learners. In addition, cooperative behaviors between homogeneous and heterogeneous robots, by performing intersection of their regular robotic languages, can be looked into as future work. Formal language theory also offers algebraic tools for analysis of the languages and automata, which can be explored for studying optimal plans of hybrid system behavior, and can aid in composing and decomposing languages.

Chapter 1

INTRODUCTION

Dynamical systems can be classified in terms of the type of equations according to which their states evolve. For example, if the state evolution is governed by continuous differential equations, we obtain a *continuous dynamical system*. Similarly, systems in which the states evolve discretely by exhibiting state jumps are called *discrete dynamical systems*. A *hybrid dynamical system* is the one in which there are co-existing continuous and discrete states. While the discrete states take discrete values, continuous states of the hybrid system exhibit both a flow along continuous equations, and also discrete jumps. The evolution of states, both continuous and discrete, of a hybrid dynamical system are governed by an interaction between the discrete and continuous dynamics components so that the two dynamics do not evolve independent on each other and are "coupled." We deal with a special class of hybrid systems in this work, describe how such interactions between continuous and discrete dynamics occur in these systems, and how such systems can be controlled. Besides dynamical equations, dynamical systems can be classified in terms of the type of their state as continuous systems, discrete systems and hybrid systems [39]. The states of a continuous system take values in a continuous space, for example a Euclidean space or a differentiable manifold, while for a discrete system, states usually take values in a finite set. According to such characterization, hybrid system is the one in which the a part of the state is continuous, while another is discrete.

Controlling a robot by manipulation of the continuous dynamical equations through control inputs might be appropriate for some robotic applications requiring

statial accuracy, such as robotic welding or assembly-line applications. The control objectives for such systems might be formulated as a (single) stabilization problem, which means that continuous dynamical control laws that make the robot's behavior meet the control performance specifications can be designed. The repetitive nature of tasks within the same environment and the availability of detailed data, enable the formulation of continuous equations that describe the behaviors for such systems, without the need for frequent alterations in the equations. However, if a robot is to carry out multiple tasks under uncertainty, it has to make autonomous decisions during execution time. In such cases robots should be able to deal with dynamic environments, ensure safety, adhere to hardware constraints and handle complex specifications. The design of continuous control objectives for such systems might not be easily formulated as a single stabilization problem. Such robots are being increasingly found in military, service and space exploration applications, that call for a flexible approach for controller design that can allow work under widely varying parameters.

One approach to designing such a flexible system is by dividing the control task into a series of subtasks performed by separate closed loop controllers, and switch among these controllers to accomplish the ultimate goal. The continuous controllers are designed as specific closed loop dynamical systems, optimized for the given sub-task. For example, a controller may steer a robot between waypoints, another might collect measurements, while others communicate, interact with and manipulate the robot's physical environment. Though some basic tasks can be performed through the execution of a single controller, many tasks of interest require the system to combine different actions and switch between available controllers. Discrete switchings between these modes of operation enable a much richer behavior and make the system versatile.

The discrete switchings in such systems do not occur arbitrarily, but are

dependent on the continuous dynamics. For example, if a continuous controller governing a limited-range infrared sensor detects an obstacle in the robot's navigation path, it might trigger a discrete switching, which in turn resets the system's current dynamical equations to another set of dynamical equations optimized for obstacle avoidance. Such systems using continuous closed-loop controllers that interact with discrete dynamics of the system, define a special type of hybrid dynamical systems in which there are no resets on continuous states. The continuous dynamics of these hybrid dynamical systems affect which discrete transition can take place, while the discrete transitions reset the continous parts, possible to an entirely different set of closed loop equations.

1.1 Motivating Example

Consider a mobile robot with an on-board manipulator that needs to fetch a print-out from a printer. To accomplish this task, the robot should move from its current initial configuration to the vicinity of the printer, reach out and pick the stack of paper sheets on the output tray using the manipulator, navigate to the user who initiated the command, and deliver the printout, again using the manipulator. In the unlikely case that a single controller for the continuous dynamics can be found and shown to render the overall dynamical system stable, convergent and provably algorithmically complete, such a controller performing this function will probably be very complicated. A more natural approach is to decompose this task into a series of simpler subtasks, develop controllers for each of these subtasks, and then integrate them into a closed loop hybrid system that transitions between controllers to complete a succession of subtasks. Along these lines, the robot would first use a navigation controller to move into position to access the printer. A common PID controller could then be used to steer the robot's arm to reach out and grasp the printout. Finally, the robot would activate the navigation controller again to go

to the user who requested the job, and control its arm once again to hand out the printed sheets.

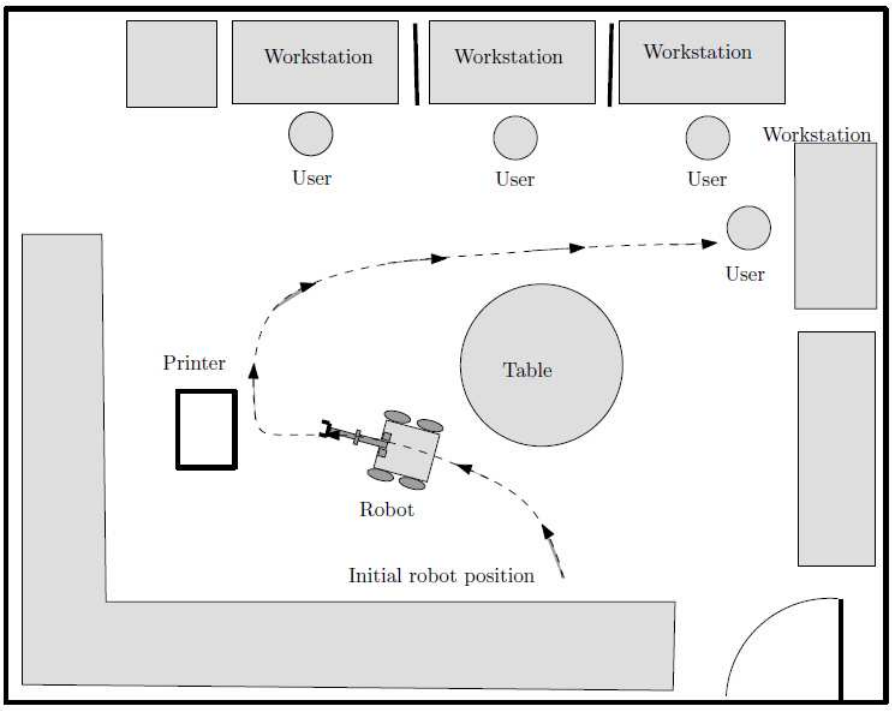


Figure 1.1: An example scenario with a robot asked to fetch a printout. This task requires the system to execute a series of basic controllers. Successful task execution depends on the ability to sequence the transitions of the control law switchings in a timely manner. The overall system is thus hybrid in nature.

Assuming that there exists a navigation controller that enables the robot to move from point A to point B, and PID controllers to move its onboard manipulator along a specified workspace path to grasp and release objects, the overall dynamics are expressed as a sequence of discrete switchings between continuous controllers. Switching to a controller is not arbitrary, but is enabled only when certain mathematical and logical conditions (will be later denoted the Pre conditions) required for the controller to run are fulfilled. Executing the continuous controller then alters these conditions and leads the system to a new state that now satisfies another set of

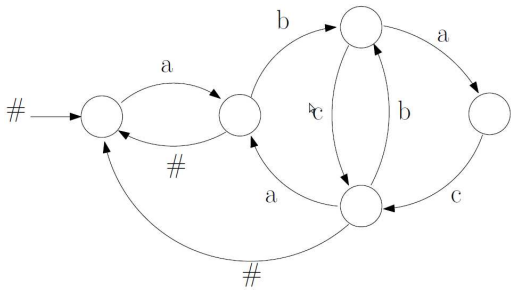
mathematical and logic conditions (denoted POST conditions), from which another switching may be possible if the current state enables the PRE conditions necessary for that other controller to be fulfilled. The discrete dynamics are therefore dependent on the continuous dynamics, and because with each discrete switching a new set of continuous dynamics is enabled, the continuous dynamics are also dependent on the discrete dynamics. Such behavior makes the system hybrid.

For the hybrid system to be able to switch between different controllers, we need to ensure that such a switch will result in predictable behavior and is permitted by the underlying dynamics. One way to guarantee this is by requiring the states reached upon running a controller to satisfy the PRE conditions of the next controller scheduled to be activated. Such guarantees can be provided by assuming that the execution of a controller will cause the system's states to evolve to some positive limit set, and all states of that set satisfy the POST of the other controller. Such constraints give us a special class of hybrid systems with assumptions on the attractiveness of each controller's limit set. There are no resets in the continuous states of the hybrid systems, but only a switching of controllers that drive the current states to states that satisfy their POST conditions.

The switching behavior between continuous controllers can be described by means of a graph that contains nodes or vertices, representing the states of the system, and arrows connecting the vertices, representing execution of each controller. Such a graphical representation with states as nodes and arrows between states is commonly used in the theory of computation and formal languages to describe discrete models of computation. Such graphs are called *automata* and if there are finite number of states in the graph, we refer to them as *finite state automata*. The finite state automaton for the hybrid system described in the last paragraph is shown in figure 1.2(b), where the transitions between states are labeled by the robot controllers and the nodes represent different states that mathematically capture the

states of the robot relative to the environment.

(a) A robotic mobile manipulator.



(b) The finite state automaton that accepts only the robot's admissible behavior.

Figure 1.2: A hybrid robotic system with three basic controllers: navigation controller a; controller for picking an object b; controller for placing the object c. The symbol # indicates the beginning and end of a motion plan.

To clarify some of these ideas, we consider a kinematic model for the robot of figure 1.2(a) that can be realized as follows. Let us assume, that the robot dynamics are given by

$$\underbrace{\begin{aligned}\dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega\end{aligned}}_{\text{base}} \quad , \quad \underbrace{\begin{aligned}\dot{x}''_e &= u_1 \\ \dot{y}''_e &= u_2 \\ \dot{z}''_e &= u_3\end{aligned}}_{\text{arm}} \quad . \tag{1.1}$$

The platform and arm of the robot are given velocity references as inputs: (v, ω) are the linear and angular velocity for the base respectively, and (u_1, u_2, u_3) is the end-effector velocity vector. The coordinates of the point on the platform (base) where the manipulator is attached is $q_p \triangleq (x, y)$, while θ is the orientation of the base. The cartesian coordinates of the robot's end effector, expressed in the body-fixed coordinate frame can be expressed as $q_m \triangleq (x''_e, y''_e, z''_e)$. These coordinates are local to the robot and are relative to the point where the arm connects with the base. This body-fixed frame is aligned with the base's orientation θ. We write

the quantities with a double prime (for example, a'') to denote the value of the quantity in the body-fixed coordinates, as opposed to writing a which is means the value of the variable expressed in the global coordinate frame. The configuration space for the system is then a sixth dimensional manifold $\mathcal{X}$ consisting of the tuples $(x, y, \theta, x''_e, y''_e, z''_e)$. We can construct a basic navigation controller $u_a = (v, \omega)$ to drive the robot from any initial point q_p to any other desired configuration q_p^d in the obstacle-free workspace as follows:

$$v = -\frac{\partial \varphi}{\partial x}\cos\theta - \frac{\partial \varphi}{\partial y}\sin\theta, \qquad \omega = \vartheta - \theta,$$

where $\varphi(x, y)$ is a navigation function [53] —a special artificial potential function without local minima— and ϑ is the angle of its negated gradient.

As mentioned before, for steering the end-effector (gripper) to a desired location, a series of PID controllers typically found in most commercially available manipulators, can be used for the arm. The input to these controllers is the coordinates of the point $p_0 = (x''_0, y''_0, z''_0)$ to which the end effector must be steered. The PID control then steers the arm to this point so that $q_m = p_0$. Then, elementary arm motions can be sequenced to achieve simple pick and place tasks as follows: to pick, steer the end-effector to the location of the object by specifying the point at which the object is placed, close the gripper to grasp the object, and return the arm to a home position; to place, move the end-effector from the home position to the desired object location, release the object by opening the gripper, and return the arm to the home position. Of course, numerous issues with regards to sensing, feedback, robustness, and repeatability can arise, however these are beyond the scope of this work, and are rarely addressed in the simple control interfaces provided by robotic arm manufacturers.

Summarizing the controllers that the robot has, we deal with three basic controllers on the robot, each one specialized in a sub-task:

- u_a: a navigation controller for the mobile platform that moves the robot base to a destination point q_p^d from any initial obstacle free configuration;
- u_b: a controller for the manipulator that steers the arm of the robot to pick an object from location q_o and hold it in a pre-specified "home" position q_m^h;
- u_c: a manipulator controller that releases an object from its gripper at a desired location.

To complete a task that requires remote manipulation and transport of objects, we need a plan that would properly sequence these continuous controllers through discrete switchings.

There could be alternative ways to design a navigation controller, for example, those based on computer-vision algorithms. Similarly there could be alternative controllers for the arm. One of the focus of this work is to capture the behavior of such controllers with a model that would allow planning of more complex tasks through their automated sequencing. We do not deal with designing or improving such controllers, but only use the ones already designed, verified and available for use to do planning and control and achieve goals through proper controller switching. Our approach however assumes that the available controllers *converge* the system's state to the desired specification points or sets in the state space. These limit sets might satisfy the pre-conditions of more than one controller and therefore the system can switch to either depending upon what post-condition will ultimately satisy the specifications. For example, there must be guarantees that the navigation controller steers the robot to the vicinity[1] of a desired point, specified by its coordinates in the robot's workspace in finite time. Similarly, the controllers for the arm should be able to steer the end effector in the vicinity of the desired goal point in finite time.

[1] By vicinity, we mean that the robot should be reasonably close to the desired location, upto within an allowable error decided by the system's designer.

When such guarantees are provided, the design of the controllers does not interfere with the planning problem.

The concept of convergence is closely related to *stability*. Informally, a system's dynamics are said to be *(Lyapunov) stable* if the states of the system remain close to an equilibrium point or a positive limit set in the state space, whenever its states start close to the equillibrium point or set. For a physical system, stability refers to the system remaining within safety limits (bounds) defined due to hardware, logical and mathematical constraints. For example, the robot's velocity must remain within operation speed limits set by the manufacturer. *Convergence* refers to a system's state approaching an equilibrium point or a limit set as time progresses. It is possible that the states of the system do not converge to a limit set, but remain close to it and hence the system being stable but not convergent. When a system is stable and convergent, it is said to be *asymptotically stable*. If we have controllers that provide assurance for stable and convergent behavior, we can ensure that running a particular controller will steer the robot to a particular limit set in the system's state space and stabilize the system at the state until a new set of controllers stabilizes the system to its own limit set. This limit set can be mathematically encoded in terms of the conditions that are satisfied upon running each controller called the *post-conditions* of the controller. In order to run each controller, however, a set of *pre-conditions*, encoding the set of parameters needed for a controller as input, must be satisfied. Provided the pre-conditions of a controller are met, stability guarantees that the controller will lead the system to its set of post-conditions. This is a useful fact since now, if the pre-conditions of a controller, say f, are satisfied by the post-conditions of another controller, say e, the system can switch from controller e to controller f. The system then flows from a state that satisfies the pre-conditions of the controller e to a state satisfying the post-conditions of e. At this point, assuming that the pre-conditions of the controller f

are satisfied, switching from controller e to controller f occurs and the continuous dynamics is reset. Finally, the states evolve along the dynamics of controller f to reach a state that satisfies the post-conditions of controller f.

This work is not concerned with a thorough characterization of answering how to guarantee stability or convergence mentioned in the previous paragraphs, but rather assumes these two properties for the low-level, closed loop dynamics, and goes on to investigate how the hybrid robotic system can be mapped into an automaton like the one shown in figure 1.2(b). The process of mapping a detailed hybrid dynamical system into a purely discrete computation model like the finite state automaton of figure 1.2(b) is known as (discrete) abstraction. As we shall see, such mapping enables us to perform planning by considering a purely discrete system. Since the states in a discrete system only take a finite number of values as compared to infinitely many values that the states of a continuous system can take, checking properties, such as ensuring the states evolve only in the "safe regions" of the state space, for different states of a discrete system only requires a finite number of steps as compared to checking them on a continuous system.

1.2 Abstractions of Hybrid Systems

In order to solve the planning and control problem on a hybrid robotic system with stable continuous dynamics, it is necessary to determine what we can expect out of the future evolution of the continuous dynamical equations. This typically gives rise to reachability computations. Checking for reachability on a continuous dynamical system requires solving the continous equations for all the initial states that the system may possibly take. Since the states of a continous dynamical system can take infinitely many values, reachability analysis for a continuous dynamical system becomes computationally intractable. Discretizing the state space offers a solution by enabling checking of specifications on a finite number of locations. In the worst case, this would require exhaustive checking of all possible cases, though

at least the number of locations is finite. For example, in the example of section 1.1, to check whether the system leads to a particular equilibrium set from a given set of initial conditions, the continuous dynamical equations (1.1) will have to be solved. The process needs to be repeated for every set of initial and final conditions. However, if there are stability guarantees of the system's continuous equations, we might be able to "predict" accurately, where the system will end up starting from a given set of initial conditions, without having to solve the continuous equations (1.1). Stability enables us to discretize the system, for example to a discrete finite state automaton of figure 1.2(b), by only accounting for the controller index being activated, along with possible controller switchings, while disregarding the continuous dynamics involved behind execution of each controller. We are therefore able to reduce the complexity associated with reachability calculations.

Such process of mapping a detailed hybrid dynamical system into a purely discrete computation model like the finite state automaton of figure 1.2(b) is known as (discrete) abstraction. Abstraction facilitates automated planning and control by reducing the complexity of the interacting continuous and discrete dynamics of the system. Planning and control is then performed on the smaller system and the results can be implemented on the original system. For abstractions to be of value, they must relate the larger system and its abstraction through a relationship that establishes an equivalence between the two systems, since otherwise, there is no guarantee that a plan charted on the abstract system is implementable on the original hybrid system. Such an equivalence relationship partitions the continuous state space of the system and groups together states that behave in a "similar" manner. This leads to discretization of the system where the infinite values that states can take are reduced to a finite number equal to the number of equivalence blocks formed by partition. Then, instead of tracking the behavior of the system from each continuous state, one can track the behavior of whole blocks.

The equivalence between systems is usually established by using two general approaches. One of them is the partition-based approach in which the state space is partitioned on the basis of one or some of the properties of the system that "we care about." The properties used in partitioning could be spatial, logical or mathematical properties of the system and all the states that satisfy a desired property are grouped together into one block of the abstract system. The analysis is then performed on the partitioned (quotient) system instead of the original system. In the second approach, equivalence is sought based on the evolution of the states of the system. All the states that evolve "similarly" according to the system's dynamics are grouped together as blocks and the evolution of these blocks is then observed instead of the observing the behavior of individual states. All the equivalent states grouped in one block can evolve to the same partition blocks. The two systems are then said to be related by a *bisimulation* relation ([45]), in which the abstract system can *simulate* the dynamics of the original system and vice-versa. A similar but weaker notion to that of bisimulation is *simulation*. One system is said to *simulate* another system if the behavior of the former is matched (*simulated*) by a behavior of the latter system, but not vice-versa [45]. Thus simulation is in some sence a one-way relationship between two systems whereas a bisimulation establishes a two-way relationship.

In this work, we use a particular class of hybrid systems that allow for a new type of abstraction which combines the two approaches of abstractions discussed in earlier paragraphs. These hybrid systems have a set of asymptotically stable continuous dynamics with parameterized attractors. By means of the parameters of the hybrid system, that determine the location of these attractors, the designer exercises control over the hybrid system forcing certain guards to be activated and thus determining the macroscopic reachability properties. Switching between different continuous dynamics (modes) is not arbitrary; for a certain mode to be activated

certain conditions have to be met, and similarly, steady states of continuous dynamics satisfy given specifications. We mathematically characterize and computationally approximate these pre– and post-conditions without resorting to reachability calculation. Control is exerted on these hybrid systems by first choosing among a finite set of closed loop dynamics, and then setting the parameters of this dynamics to determine the location of the attractors and steer the trajectories to a particular set and activate the associated guard. Control decisions are hybrid because they involve the discrete choice of a closed loop vector field, and the selection of an attractor in a continuous state space. We show that this class of systems admit an observable bisimulation that gives rise to a discrete transition system which represents faithfully the vector field transitions and the asymptotic behavior of the hybrid system.

The abstract transition system is constructed using characterizations of attraction regions and limit sets of the continuous dynamics of the concrete transition system, for which stability certificates are assumed. Such certificates can be expected to have been constructed during earlier phases where the parameterized continuous controllers are designed for the hybrid system. In this work, instead of constructing the stability certificates, the focus is on constructing the discrete abstraction *after* the stability certificates are provided, and establishing the behavior equivalence between the concrete system and its abstraction which allows the design of high-level planning strategies with guaranteed low-level implementation on the concrete hybrid system. Such work does not suggest ways for controlling the continuous dynamics of a hybrid system, but rather provides models that capture the abstract behavior of the closed loop hybrid system, given these continuous controllers. In other words, we do not indicate how a particular system should be controlled at the low level, but propose ways to organize existing, established low-level control strategies to give rise to more complex high-level behavior.

1.3 Parallels Between Robotics and Formal Languages

Controller sequencing in the particular class of hybrid system we consider is possible by checking compatibility between the pre-conditions specified by the parameterized domains of the controlles and post-conditions captured in the limit sets. All acceptable sequences of controllers form the *language* for the system. The strings of controller labels (symbols) in this language can be shown to be accepted by the finite state automaton, abstracted from the hybrid system. For example, refering to the automaton of figure 1.2(b), the string of controllers $a\,b\,a\,c\,a\,b\,c$ represents a valid controller sequence for the hybrid system of the robot of figure 1.2(a), and therefore is also in the system's language.

The study of control action strings is not unique to the hybrid systems considered in this work but is also found in other versions of discrete planning and control of robots, in the context of symbolic control [4]. Symbolic control involves control of systems using strings of symbols, which for our case represent stable controllers, to achieve the overall goal. In order for a system to achieve an overall behavior, the symbols in a string must be connected to each other using some logical or mathematical conditions. These conditions naturally give rise to constraints on the concatenation rules for strings. These constraints usually fall under two categories: local constraints that affect the next symbol for a controller in a string, and long-distance constraints in which the appearance of a symbol in a string depends upon a preceding symbol, with these two symbols separated by a substring of other symbols in the sequence. An example of local constraints could be the concatenation rule for the example of section 1.1 in which two symbols exhibit local dependencies owing to the pre– and post-conditions compatibility. In the same example, we could see *forbidden strings* such as $b\,a\,a\,b$ which represent a controller sequence that asks the robot to pick up an object, navigate to two points using controllers a, and then

try to pick up another object using a b controller which is impossible since the gripper can hold only one object at a time. This string exemplifies the long-distance behavior between two b controllers, that are long distance separated by controllers with label a.

Local and long-distance dependencies are also found in patterns of sounds in some spoken languages [54, 55]. Studies on these constraints help phonologists understand the properties of languages and therefore predict, model and study the evolution of speech patterns of these languages. The theories developed by phonologists in this area enable us to apply them to our symbolic control for analysis, planning and control of systems that generate such patterns. Furthermore, the concept of languages and automata are inter-related since there is a language generated by each automaton, and given a language, there are procedures that allow construction of automata that recognize such a language [29,57]. Therefore, a wide realm of computational tools developed in computer science for the study of formal languages are available for symbolic control.

In formal language theory, languages are characterized on the basis of subset relationships of language inclusions that form a hierarchy shown in figure 1.3. This hierarchy is called the Chomsky hierarchy [9,10]. Each language class in the hierarchy has specific algebraic properties with associated expresivity and computational complexity. In general, languages higher in the hierarchy can express more complicated patterns, and therefore include the ones lower in the hierarchy. However, such increased expressiveness comes at an increased complexity of the automata describing the languages [30], and therefore an increased cost of recognizing the languages using automata, or performing boolean operation on languages. The focus in this work is to find a language class that offers relatively cheap ways of representing the languages that we see in symbolic control sequences, while being reasonably expressive to capture the local and long-distance constraints that govern the generation of

these languages.

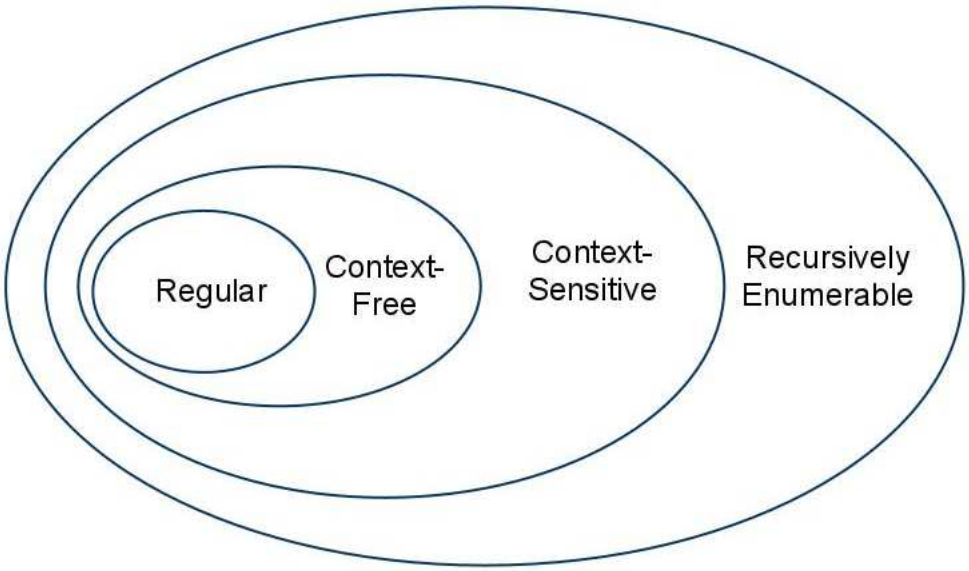


Figure 1.3: The Chomsky hierarchy of language inclusions.

In addition to characterizing complexity and computational costs of language operations, formal language theory also offers answers to questions such as whether there exist algorithms to recognize a particular language, whether a particular string is in a given language, or whether two descriptions of language actually describe the same language. When there are known algorithms that can provide a definitive answer to these questions in a finite number of steps, we say that these problems are decidable. Languages lower in the Chomsky hierarchy offer benefits over the classes higher in the hierarchy because more problems related to language operations are decidable [29]. Though regular languages are the lowest class of languages in the Chomsky hierarchy, there are even smaller sub-classes of languages forming another hierarchy known as Sub-regular (figure 1.4) [54,56]. Sub-regular languages, in addition to offering all benefits of regular languages in terms of computational

complexity, also provide a narrower domain of properties specific to each sub-class. These sub-classes capture more accurately the interaction of phonological sounds when producing words that would be considered correct in some natural language. Local constraints impose conditions on contiguous sequences of language symbols, and generate languages that belong to the *local branch* of the Sub-regular hierarchy. Likewise, long-distance dependencies impose conditions on subsequences of language symbols, not necessarily contiguous, and generate languages that belong to the *piecewise branch*. With the constraints found in robotics exhibiting local and long-distance constraints, similar to what has already been studied in computational linguistics and phonology. An exploration in this area from a dynamical systems perspective, therefore promices to identify computational tools that facilitate the analysis of languages that can describe sequences of robotic controllers.

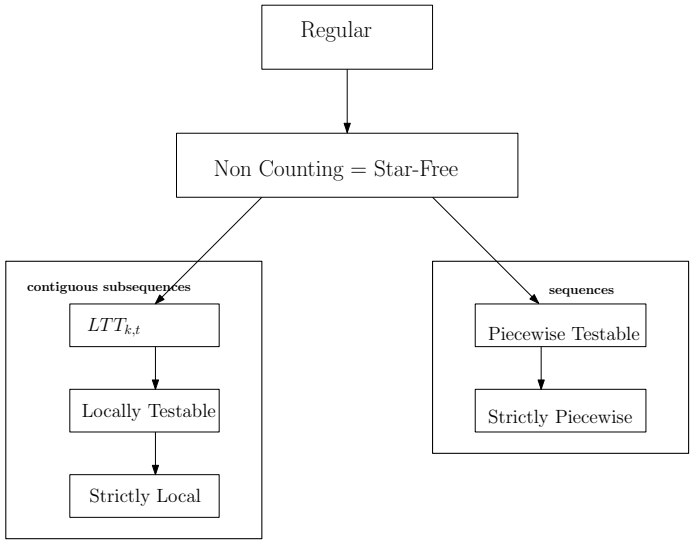


Figure 1.4: The Sub-regular hierarchy of language patterns.

1.4 Basic Hypothesis

In summary, we find that working with hybrid systems with stable continuous dynamics enables us to discretize the system and deal with a transition system

that retains information about possible controller switchings. Such discretization becomes possible due to stability guarantees that help us establish equivalence blocks in the state space of the hybrid system, based on the limit sets to which these states stabilize. Two controllers can be sequenced if the post-conditions of one controller, determined by the limit set to which the controller stabilizes, match the pre-conditions of the next.

The transition system provides information on possible controller sequences that a robot allows execution for. Therefore, instead of looking into the dynamical equations governing the evolution of states in the hybrid system, the transition system allows higher-level planning by focusing on which controller must follow which, in order to accomplish some task. Implementation of the plan generated by transition system is guaranteed on the hybrid system due to equivalence established by the abstraction procedure.

We will see that our hybrid systems can be abstracted to discrete transition systems that can be represented as a type of automata. These automata define languages formed from an alphabet of controller symbols, which are concatenated based on mathematical rules, local and long-distance constraints to yield words. Thus our study allows the use of formal language theory to analyze systems that generate strings of languages that obey local and long-distance constraints.

1.5 Contributions

The starting point of the work in this thesis is an abstract representation of a particular class of hybrid systems with convergent continuous dynamics and no resets in continuous variables. We establish equivalence between the abstract representation and the underlying hybrid system through a (weak) bisimulation relation. The equivalence enables us to focus on planning the system's overall behavior through controller sequences observed in the abstract system, instead of dealing with the dynamics behind each controller of the hybrid system. We study the languages

generated by the sequences of controllers and identify their place in the Chomsky and the Sub-regular hierarchies. Based on the benefits identified in finding the lowest classes in the hierarchies, we aim at characterizing the lowest class that would effectively capture the constraints of our languages, within the Sub-regular hierarchy. More specifically, we show that the local and long-distance constraints generate sub-regular languages, which when intersected together, give us a new class of sub-regular languages. This new class can describe the behavior of robots that can be modeled as hybrid systems with asymptotically stable continuous dynamics and no resets on continuous states.

Chapter 2

LITERATURE REVIEW

Careful attention to a system's stability is required for the control of continuous closed-loop dynamical systems, especially when the systems are non-linear [7,32,46]. Control of hybrid systems requires even greater attention, since switching between stable continuous controllers can destabilize an otherwise stable dynamical system [12,49]. For hybrid and switched systems, it has been shown [27] that restrictions must be placed on the structure of the system, as well as on the switching signals, in order to make the system stable or convergent to a limit set. In many cases, the original switching signals of the hybrid system must be modified in order to stabilize the system [38,48,49].

Even if a hybrid system can be stabilized, there is significant computational overhead involved in checking the system for *reachability* [20,59,66]. Reachability refers to finding states that a system can reach within its state space. We must be able to *verify* that the system avoids "unsafe regions" of the state space, regions that might violate the system's constraints or control objectives. Owing to the continuous dynamics of the hybrid system, the hybrid states can reach an infinite number of locations in the state space. Calculating all the (infinite) reachable points for the system's verification is extremely computationally demanding.

Several issues related to stability and reachability in hybrid systems may be resolved by working with purely discrete equivalent descriptions of these systems, which provides motivation to the use of symbolic control. Checking for reachability

in finite discrete dynamical systems, in the worst case, involves exhaustive checking of all the (finite) states that a system can be found in. Thus, reachability calculations for discrete dynamical systems are at least computationally tractable compared to those for hybrid dynamical systems. Similarly, discrete systems also present a computationally cheaper stability analysis since the states only need to be checked for and controlled within a finite number of values. To capitalize on the advantages offered by discrete dynamical systems, continuous and hybrid dynamical systems are abstracted to a discrete system and analysis is carried out thereafter on the discrete system [4, 5, 60]. The discretizations simplify the domain of application of differential equations by breaking up the workspace or controller dynamics into simpler components, which when appropriately sequenced together yield a behavior that satisfies certain specifications.

2.1 Discrete Models of Hybrid Systems

The need for reigning in complexity is apparent in model checking and verification of general classes of hybrid systems [23, 51], particularly when direct reachability computation is involved [20, 59, 66]. Abstraction offers a way to facilitate computations by enabling the analysis to be performed on a smaller system, and allowing the generalization of the conclusions to the larger system.

Working with purely discrete models based on finite state automata helps alleviate the problem of reachability computation, since in the worst case, the whole (finite) state space can be searched. One way of obtaining discrete models for hybrid systems is through abstraction. Abstractions are consistent when they preserve properties between two models of different size by establishing equivalence relations between the states of each system. The relation that links states across different systems naturally induces partitions in the state space of the larger system, by grouping together the states that are related to the same state on the smaller system. When the equivalence relation is a *simulation* or a *bisimulation*, each equivalent block in

the partition contains states of the system that evolve according to the continuous dynamics in a "similar" fashion. This enables the analysis of the system by considering blocks of equivalent states, rather than the individual states themselves.

One commonly used equivalence relation that establishes a two-way equivalence between systems of different size is bisimulation [43]. States across two bisimilar systems are related by an equivalence relation such that the evolution of states in either system can be matched by a "similar" evolution of equivalent states in the other system. However, establishing bisimulation relations becomes too demanding [2] and for most general classes of hybrid systems, the procedure that is supposed to create the partitions corresponding to a bisimulation, is undecidable. *Decidability* refers to a computational or algorithmic procedure, which, given a hybrid system and a desired property, will verify in a finite number of steps whether the system satisfies the property or not [2]. The survey paper [2] establishes boundaries on decidability of systems abstracted using bisimulations and shows that one has to severely restrict either the continuous dynamics of the system [1,50], or the discrete dynamics [37] in order to be able to abstract the system using a bisimulation.

To be able to use a simpler system for the analysis of a more complex hybrid system, instead of hunting for a two-way relationship, a less restrictive equivalence relation known as simulation [43, 44] might be appropriate. Simulation provides a one-way equivalence between systems and can preserve some desired properties that the designer cares about when compressing the system's size, while abstracting more information than a bisimulation. When a system is abstracted by means of a simulation relation, a property verified for the abstract system will hold for the original system. The converse, however, is not necessarily true, as would have been the case if the equivalence established was bisimilar. Abstractions based on simulation relationships were described in [63], in a framework similar to that of a bisimulation in [47].

Approximate simulation and bisimulation results for discrete abstractions also exist [19, 61]. Approximate bisimulation [18] allows the trajectories of the two related system under the abstraction map to be "close" in a Lyapunov-like sense, instead of being matched exactly as required by bisimulation. There are many approaches for discrete abstractions found in literature that fall under two main categories. One is based on simulation, bisimulation and their approximate counterparts, while the other is based on partitioning the state space [3, 22, 65]. Discrete abstractions allow the use of symbolic control to be applied to hybrid systems by considering the discrete partitions as symbols. Planning and control then involves deciding how and when to switch between symbols, and this gives rise to *symbolic control*.

2.2 Symbolic Control Synthesis

Discrete abstractions for hybrid dynamical systems usually follow a hierarchical design that extends from the user interface at the highest abstract level to robot hardware control at the bottom level (figure 2.1). Broadly, there are three levels in the design hierarchy [4]:

1. At the top is the *specification level* that defines the overall behavior of the system through a directed graph such as a finite state automaton. All paths in this graph satisfy the specification (they obey all constraints) for the system and correspond to valid symbol sequences that can be executed on a system.

2. The middle level is the *execution level* at which the individual paths of the graph are designed according to constraints between each controller or symbol.

3. The paths are implemented on an actual system at the *implementation level* in which the symbols correspond to the actual system dynamics.

Depending on whether the designer builds the system starting with design specifications and adapting the specifications to the implementation, or whether he builds

the system by chosing existing controllers in the implementation level and then abstracting the controller dynamics to generate valid paths and specifications, the design methodology can be classified as being top-down or bottom-up, respectively.

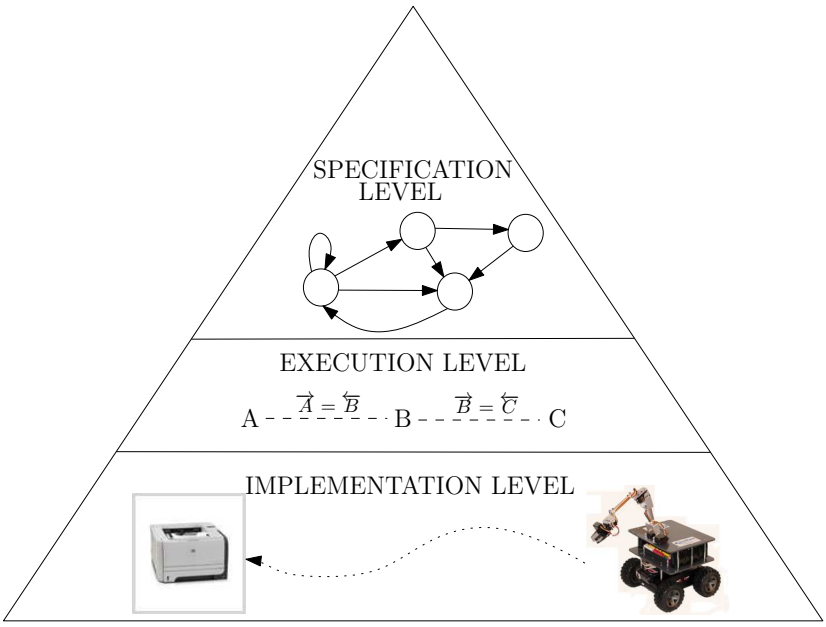


Figure 2.1: Figure showing the hierarchical structure in symbolic control designs, as it would apply to a "fetch-a-printout" example of section 1.1. The specification level describes the overall abstract behavior of the system; the execution level describes feasible sequences of symbols (plans) depending on whether the post-conditions ($\overrightarrow{\cdot}$) of one symbol satisfy the pre-conditions ($\overleftarrow{\cdot}$) required by the following symbol; the implementation level executes the plan on an actual system.

There are several approaches found in literature for a system's discrete abstractions. All of these approaches can be broadly classified into two categories – environment driven strategies and control driven strategies [4]. We will first discuss these strategies for discrete abstractions in sections 2.2.1 and 2.2.2. Top-down versus bottom-up design synthesis will be discussed next. As we will see, current system designs typically follow a top-down approach. We will examine the benefits

and limitations of each approach and finally examine the steps that can be taken to overcome some current limitations.

2.2.1 Environment-driven Strategies

Environment driven discretization involves partitioning of the system's workspace. This is usually done by discretizing the workspace using polygons of appropriate shapes [5, 11] that enable construction of stable control objectives for each polygon. System control strategies evolve within each polygon and the system then "flows" between adjacent polygons to reach the desired goal. The work in [11] describes such a control strategy with the workspace discretized into triangular areas. Although, the work in [11] does not explicitly deal with symbolic control, it does deal with discretizing the workspace to allow control by sequencing the flow between adjacent polygons. Such discretization allows the polygons to be considered as symbols and thereafter perform planning.

The polygons in [11] are chosen by discretizing the free space of the environment. Such a design circumvents the control problems [33, 35, 36] related to navigation in cluttered environments. Simpler polygonal regions, also allow complex control tasks to be described by using logic statements for directing the flow between these regions. Linear temporal logics [14, 34, 62] use such a control strategy by allowing one to specify the desired temporal behavior of the robot by using operators like "next," "until," and "always." In this way, one can encode control objectives that involve prioritizing and sequencing of tasks.

Since environment driven methods are environment specific, when presented with new environmental settings, the abstraction procedure must be repeated. This involves defining new polygons, describing a new set of control objectives for each polygon, and designing new control strategies for the flow of the states between these polygons. Such a requirement presents a major limitation to the use of these strategies unless the system is to be used in a dynamically static environment.

Moreover, techniques based on environment driven discretization have to rely on model checking algorithms [17,28] to ensure that the design conforms to the desired specifications and is implementable on an actual dynamic system. Model checking involves an exhaustive checking of the system's state-space and becomes inefficient for systems with large state spaces [41].

2.2.2 Control-driven Strategies

Whereas environment driven strategies discretize the systems' work environment, control driven strategies discretize the continuous dynamics of the system. For example, in the case of a robot navigation problem in which a robot starts at a point in its workspace and navigates to a set goal point, the overall navigation trajectory can be discretized using motion primitives like "go forward," "turn left," "turn right," etc. Execution of these motion primitives in an appropriate sequence and with the right duration, may steer the robot to its goal.

The technique of discretizing the robot's behavior in terms of its available control actions and their associated closed-loop dynamics has been used in the framework of motion description languages (MDLs) [6, 13, 40] and in an automata based framework in [15, 16]. Both these approaches use motion primitives as symbols. Motion description languages compose motion primitives based on time and sensory data triggers to develop behaviors. In maneuver automata, controllers are sequenced by checking the compatibility between the states reached by running a particular controller and the initial state requirements of the following controller [16].

Control driven strategies are capable of handling dynamic environments by using sensor driven switchings in response to the environmental changes. However, motion description languages do not specify a procedure to generate a controller sequencing plan from a given set of primitives, a robotic system and constraints between the primitives [40]. The design of such a plan involves intervention from

the designer and largely depends on the task. Furthermore, a formal basis for design of each controller has not studied in control driven approaches.

2.2.3 Top-down vs Bottom-up Design

The approaches using temporal logics and motion description languages are mostly top-down whereby specifications are designed first, followed by planning the behavior and ultimately executing the plan on the system. Systems designed by this approach can usually handle complex specifications by breaking them down into discrete sub-tasks carried out in sequence. For example, work on linear temporal logics aims at using human language-like specifications. However, top-down designs tend to over-simplify the low-level physical dynamics in an attempt to achieve compatibility with the designer's choice of high level specifications. Therefore, plans generated for controller sequences might sometimes be infeasible if certain aspects of the concrete continuous dynamics have been abstracted away.

A bottom-up design, on the other hand builds upon existing controllers from the execution level. This could be desirable for practitioners since they can use the controllers that they have designed, tested and rely upon from experience. A bottom-up approach does not require a redesign of controllers and enables automatic design synthesis. Because the design uses an existing control system with well-tested controllers, the plans generated by using abstractions of these controllers are automatically executable on the underlying system, provided the conditions necessary for activation of each controller are satisfied.

2.2.4 Automatic Design Synthesis

With most current approaches being either control or environment driven, there is a need for a design that combines the advantages of both these strategies. Such a system would be able to capture environmental constraints, handle dynamical environments, and provide a universal solution that is stable for different tasks and

environments. Furthermore, a bottom-up design is needed that would enable the use of well tested and optimized controllers already existing on a system. Using existing controllers ensures that the plans generated by their discrete abstractions are implementable on the original system, provided the plan takes into account the conditions necessary for execution of each controller.

2.3 Language Complexity

Informally, formal languages contain strings of symbols, for which there is an algorithm that can decide whether the strings are "admissible". Characterizations of these languages within the Chomsky hierarchy is useful since it identifies a domain of applicable computational and algebraic tools to analyze the languages. Hybrid systems, among many other models of computation, can be thought of as a dynamical process that generates admissible strings of symbols that define languages. The fact that a hybrid system can be thought to generate (or accept) strings of symbols, can be seen, for example, from the abstract representation of the hybrid system such as the finite automaton representation of figure 1.2. For example, the hybrid system represented by the automaton of figure 1.2 accepts the controller sequences (or strings) $a\,b\,a\,c$ and $a\,b\,c\,a$ that belong to the system's language. Toward characterizing the domain of languages obtained by abstract models, it has been shown [31] that certain variations of motion description languages are not regular languages. Furthermore, it has been proved in [31] that extended motion description languages are context-free. Linear temporal logics, on the other hand, are accepted by Büchi automata [17,58], a variant of the finite state automata that accept strings of infinite length. The languages that Büchi automata accept are called ω-regular languages [67].

While Büchi automata evolve along loops while tracing infinite words, pushdown automata (that generate context-free languages) include a stack. These characteristics make the two models expressive but also more complex as compared to

finite state automata. Though the languages they generate are richer than regular languages accepted by finite state automata, several decision problems on these languages are significantly more difficult to solve. For example, there is no algorithm that can answer in finite number of steps, for all inputs, whether two context free languages are equal, or whether their intersection is empty. Thus these problems are undecidable for context-free languages, whereas it has been shown [57] that they are decidable for regular languages. Besides advantages in solving decision problems, regular languages also offer closure properties on more operations as compared to closure properties of context-free and $\omega-$regular languages. For example, whereas context-free languages are not closed under intersection and complement, regular languages are. Also, Büchi automata are not closed under determinization [67], which means that a non-deterministic Büchi automaton cannot be converted into an equivalent deterministic Büchi automaton that accepts the same language. Non-deterministic Büchi automata are more expressive than deterministic Büchi automata. Finite state automata, on the other hand are closed under determinization, with both deterministic and non-deterministic finite state automata being equally expressive. Although regular languages may not be able to express all the type of specifications that ω-regular [14, 34, 62, 67] or context-free [31] languages can, they can still capture meaningful constraints and objectives. The work in [15] and [16] points out that maneuver automata accept regular languages. However, the expressive power of the resulting models of [15] and [16], when they are subject to constraints on controller sequencing, offer an unexplored area for research.

2.4 Aims and Scope

This work focuses on a bottom-up modeling of hybrid systems that capitalizes on existing robot controllers. The modeling methodology obviates the need to redesign controllers and proposes solutions for using the existing controllers to

carry out complex tasks. The key feature of this work is the exploitation of parallels between formal language theory and robot (motion) planning. We show that languages generated by discrete abstractions of our hybrid dynamical models are sub-regular. Our modeling framework is very similar to Frazzoli's maneuver automata [16]. However, this work goes a step further in defining a specific domain in the Sub-regular hierarchy, so that the well-established theories in formal languages and computational analysis can be applied efficiently.

We start with recently developed modeling formulations for hybrid systems, which are based on asymptotic abstractions. The systems considered have convergent (stable) continuous dynamics that can be abstracted to a discrete labeled transition system [64]. The abstraction captures the evolution of a stable system, from a certain set of initial conditions to a parameterized destination set, in finite time. Such *finite time abstraction* [50] guarantees that after a certain time period, initial system states in set A have "collapsed" into points in a set B. *Asymptotic abstraction* [50] is a generalization of this concept, with time allowed to go to infinity. The work in this thesis contributes to establishing the properties of the languages of the machines generated as discrete abstractions of hybrid dynamical systems representing our class of robotic systems.

We analyze robotic systems modeled as hybrid agents with stable continuous dynamics by abstracting the system to a labeled transition system that generates a formal language. A language-theoretic framework allows exploitation of existing results in formal language theory to bring in a fresh perspective for analysis of systems whose behaviors can be abstractly described as a sequence of controllers. Since controller sequences can be viewed as symbols, the analysis of collections of "admissible" strings are also of relevance in the context of formal languages, computational linguistics, and phonology.

Particularly, we show that the "regular robotic languages" generated by the

abstract systems are an intersection of Sub-regular languages, and as such they are naturally much "cheaper" to represent compared to possible intersections of context-free or ω-regular languages [69], [68]. The regular robotic languages belong to the star free (SF) class of the Sub-regular hierarchy. In the process of showing subregularity of our robotic languages, we introduce a class of formal languages, the tier-based strictly local languages (TSL) which describe local and non-local interactions of symbols in a language. Such a result is not only a contribution in the field of robotics [52], but also in the computational linguistics domain [26].

Chapter 3

MODELING ROBOT BEHAVIOR

For a continuous system, the states of the system change by following continuous dynamical equations, whereas a discrete system exhibits state "jumps." When there are interacting continuous and discrete dynamics in the system, we obtain a *hybrid dynamical system.*

The robot models that we consider are composed of multiple closed-loop continuous dynamics. For a controller to be executed, its PRE conditions, which include the set of logic propositions involving the state of the system, the input parameters, and the state of the environment, must be satisfied. We assume that the continuous dynamics within each controller are stable. Stability causes the execution of each controller to drive the robot's state to a positive limit set encoded in the logic propositions - the POST conditions of the controller. Controller switching occurs if the state of the hybrid system, that satisfies the POST conditions of the last controller, also satisfies the PRE conditions of the next controller. The PRE for a controller is evaluated after changing the parameters expressing the input references for the controller.

In this chapter, we model the robot as a hybrid system and show its abstraction to a discrete system. In the next section, we formalize the definition of a robot modeled as a hybrid system. In section 3.2 we will show how with the process of abstraction we can obtain a discrete transition system that only retains information about the index of each controller, and the associated parameters. Such an abstraction then leads to the concept of strings and languages that will be discussed in the

next chapter. To make the ideas concrete, we end this chapter with an example of a hybrid robotic system with planning applied on its abstraction in section 3.4.

3.1 Hybrid Robot Model

Definition 3.1.1 (Hybrid Robotic System)**.** *The hybrid robotic agent is a tuple:* $\mathbf{H} = \left\{\mathcal{H}, \mathcal{P}, \mathcal{K}, \mathcal{AP}, f, \overleftarrow{\cdot}, \overrightarrow{\cdot}, s, T\right\}$*, where:*

- $\mathcal{H} = \mathcal{X} \times \mathscr{L}$ *is a set of hybrid states. It is the cartesian product of a set of continuous states* $\mathcal{X} \subseteq \mathbf{R}^n$*, and a set of boolean variables* $\mathscr{L} \subseteq \{\mathbf{0}, \mathbf{1}\}^r$*,* $r \in \mathbf{N}$*, that can capture the discrete variables. Here,* $\mathbf{0}$ *and* $\mathbf{1}$ *represent false and true respectively.*

- $\mathcal{P} \subseteq \mathbb{R}^m$ *is the set of control parameters.*

- $\mathcal{K}$ *is a set of finite discrete locations. A location* $k \in \mathcal{K}$ *indexes a unique controller for the continuous dynamics.*

- f*:* $\mathcal{X} \times \mathscr{L} \times \mathcal{P} \times \mathcal{K} \to T\mathcal{X}$ *is a collection of assymptotically stable vector fields, indexed by* $k \in \mathcal{K}$ *and parameterized by* $p \in \mathcal{P}$

- $\mathcal{AP}$ *is a set of atomic propositions* (α_i)*.*

- $\overleftarrow{\cdot}$*:* $\mathcal{K} \to 2^{\mathcal{AP}}$*;* $\overleftarrow{k}$ *denotes the pre-condition (*Pre*) of the controller indexed by* k*, giving the set of atomic propositions which if true allow the execution of* k *from the state* h *with the parameter* p*. We write* $(h, p) \models \overleftarrow{k}$ *when the execution of* k *is possible.*

- $\overrightarrow{\cdot}$*:* $\mathcal{K} \to 2^{\mathcal{AP}}$*;* $\overrightarrow{k}$ *denotes the post-condition (*Post*) of* k*, giving the set of atomic propositions which are true after the continuous dynamics under controller* k *have reached the steady state. We write* $(h', p) \models \overrightarrow{k}$ *for state* $h' \in \mathcal{H}$ *that is assymptotically reached with controller* k *for the given parameter* p*.*

- *s: $\mathcal{H} \to 2^{\mathcal{P}}$ is the reset map for parameters p. It assigns to each hybrid state h, an allowable values of parameters from a subset of $\mathcal{P}$, and resets these values as the hybrid states evolve in the vector field f.*

- *T: $\mathcal{H} \times \mathcal{P} \times \mathcal{K} \to \mathcal{H} \times \mathcal{P} \times \mathcal{K}$ is the transition map for controller switching, according to which $(h, p, k) \to (h, p', k')$ iff $(h, p) \models \overrightarrow{k}$ and $(h, p') \models \overleftarrow{k'}$.*

The resets in the parameters p of the hybrid system **H**, occur when a transition of controllers occur according to T. One can think of parameters p as a controlled mechanism for setting the *limit sets* of the vector field f. There are no discrete jumps in the continuous dynamics $\mathcal{X}$ of the states h that evolve according to the system's differential equations. A state h *evolves* to another state h' along a vector f under a controller k parameterized by p, and we write $h \overset{k[p]}{\dashrightarrow} h'$. This transition occurs continuously along the integral curves of f and is not instantaneous.

3.2 Asymptotic Abstractions of Closed Loop Dynamics

The closed loop dynamics of the hybrid system of definition 3.1.1 are abstracted based on the assumption that the dynamics are asymptotically stable. We will first define an equilibrium state and asymptotic stability before discussing what role it plays in the hybrid system.

Definition 3.2.1 ([32]). *A function $f(x,t) : \mathbb{R}^n \times [0,\infty) \to \mathbb{R}^n$ (with $x \in \mathbb{R}^n$, $t \in [0,\infty)$), is said to be locally Lipschitz in the neighborhood of (x_0, t_0), if it satisfies the inequality:*

$$\|f(x,t) - f(y,t)\| \leq L \|x - y\|$$

for all (x,t) and (y,t) in some neighborhood of (x_0, t_0). The constant $L \geq 0$ is called the Lipschitz constant.

The norm $\|\cdot\|$, is any norm defined on space $\mathbb{R}^n$. We consider a continuous dynamical system defined as:

$$\dot{x} = f(x,t) \tag{3.1}$$

where $f : [0,\infty) \times D \to \mathbb{R}^n$ is piecewise continuous in time t and locally Lipschitz in x on $[0,\infty) \times D$, and $D \subset \mathbb{R}^n$ is a domain that contains the origin $x = 0$. The function $\Phi_t(x) : [0,\infty) \times D \to \mathbb{R}^n$ describes the evolution of the states x of the system (3.1) with $\Phi_{t_0}(x) = x_0$ and $\frac{d}{dt}\Phi_t(x) = f(x,t)$.

Definition 3.2.2 ([32]). *A state $x_e \in D$ of the system of* (3.1) *is an equilibrium point at $t = 0$ if*

$$f(x_e, t) = 0, \ \forall t \geq 0.$$

Definition 3.2.3 (cf. [32]). *An equilibrium state x_e of the system of equation* (3.1) *is said to be stable in the sense of Lyapunov if for each $e > 0$, there is a $d = d(e, t_0) > 0$ such that*

$$\|\Phi_{t_0}(x) - x_e\| < d \Rightarrow \|\Phi_t(x) - x_e\| < e, \ \forall t \geq t_0 \geq 0.$$

The equilibrium state is said to be asymptotically stable if, in addition to Lyapunov stability, there is a positive constant $c = c(t_0)$, such that

$$\lim_{t\to\infty} \|\Phi_t(x) - x_e\| = 0, \ \forall \|\Phi_{t_0}(x) - x_e\| < c.$$

Intuitively, stability in the sense of Lyapunov (or Lyapunov stability) implies that an equilibrium point is stable if a state starting close to the equilibrium point remains close to it forever. If in addition, the state starting close to the equilibrium point eventually reaches the equilibrium state, the state is said to be asymptotically stable. The term *asymptotically stable dynamics* refers to the closed loop dynamics

of a system such that all the states evolving according these dynamics eventually approach to an asymptotically stable positive limit set.

Definition 3.2.4 (cf. [32]). *A state (or a point) $h_j \in \mathcal{H}$ is said to be a positive limit point of $f(h_i, p, k)$ if there is a sequence $\{t_n\}$, with $t_n \to \infty$ as $n \to \infty$, such that $f(h_i, p, k) \to h_j$ as $n \to \infty$. The set of all limit points of $f(h_i, p, k)$, $h_i \in \mathcal{H}$ is called the positive limit set of $f(h_i, p, k)$, denoted $L^+(k; p)$.*

Refering to the definition of hybrid system 3.1.1, the stability of continuous dynamics means that upon running each controller, the state of the system remains bounded and it evolves toward a particular limit set (definition 3.2.4). This limit set is captured by the POST conditions of the controllers in definition 3.1.1. When such a stability property is guaranteed, we can be sure that upon running a controller k from a state satisfying its PRE conditions, the POST conditions $\overrightarrow{k}$, of controller k, are always satisfied. This is important for being able to abstract the continuous dynamics of the system, since one can switch between controllers if the POST of the current controller and the PRE of the next are compatible.

We now explain when two controllers can be concatenated. To be able to do so, we first define a distance metric for our system.

Definition 3.2.5 ([32]). *The distance of a point $x \in D$ to a subset $\mathcal{A} \subset D$ is denoted* $\mathrm{dist}\big(x, \mathcal{A}\big)$ *and is defined as* $\mathrm{dist}\big(x, \mathcal{A}\big) \triangleq \inf_{y \in \mathcal{A}} \|y - x\|$.

The post-conditions of a vector field associated with controller k can be directly related to its positive limit set:

$$\textsc{Post}(k) \triangleq \{(x, \ell, p) \mid (x, \ell, p) \models \overrightarrow{k}, \ell \in \mathscr{L}, p \in s((x, \ell)), x \in L^+(k; p)\} \qquad (3.2)$$

The pre-conditions, on the other hand, are associated with the region of attraction of a limit set $L^+(k;p)$:

$$\left\{x \mid \exists \ell \in \mathscr{L},\ p \in s\big((x,\ell)\big) : (x,\ell,p) \models \overleftarrow{k}\right\} = \left\{x \mid \lim_{t\to\infty} \operatorname{dist}\big(\Phi_t(x), L^+(k;p)\big) = 0\right\}. \tag{3.3}$$

The atomic propositions in $\mathcal{AP}$ constitute a set of logical predicates that can describe any Pre or Post of some parameterized vector field, as a well-formed formula which includes conjunction and negation operators.

In the context of this work, the following assumption is made to show when two controllers can be concatenated. The notation $\partial\{\mathcal{S}\}$ denotes the boundary of the set $\mathcal{S}$.

Assumption 1. *In the hybrid automaton* **H** *of definition 3.1.1, for all* $h \in \mathcal{H}$ *and* $p \in s(h)$ *for which there exist* $k' \neq k$ *such that* $T(h,p,k) = (h,p,k')$*, there is a constant* $d > 0$ *such that* $\forall x \in L^+(k;p)$, $\operatorname{dist}\big(x, \partial\{x' \mid (x',\ell,p) \models \overleftarrow{k'}\}\big) > d$ *and* $\operatorname{dist}\big(x, \{x' \mid (x',\ell,p) \models \overleftarrow{k'}\}\big) = 0$.

What this assumption implies is that limit sets of trajectories have to be properly contained in the Pre of a controller for the system to be able to activate this controller. For example, the robot should first move so that an object is well within the reach of its arm before it reaches out and grasps it; enabling this action on the boundary of the arm's workspace forces the arm into kinematic singularities. A consequence of the above restriction is also that although a controller may be re-parameterized on the fly (e.g., on the way to the door, decide to go the the center of the room instead), switching from one controller to another can only happen when the continuous dynamics have practically settled.

3.3 Transition System Representation

With the hybrid system defined, we now move on to defining a transition system which will serve as an abstraction for the hybrid robot system **H** of definition 3.1.1. The transition system tracks the transitions between controllers.

Definition 3.3.1 (Finite labeled transition system)**.** $\mathbf{T} = (\mathcal{Q}, \Sigma_T, \Delta_T, \mathcal{Q}_0)$ *consists of:*

- $\mathcal{Q}$ *is a finite set of states;*
- Σ_T *is a finite alphabet;*
- $\Delta_T \subseteq \mathcal{Q} \times \Sigma_T \times \mathcal{Q}$ *is the transition relation;*
- $\mathcal{Q}_0 \subset \mathcal{Q}$ *is the set of initial states.*

If $(v, \sigma, v') \in \Delta_T$ we may write $v \xrightarrow{\sigma}_T v'$. Let $\Sigma_\tau \subset \Sigma_T$ and call the transitions $(v, \sigma, v') \in \Delta_T$ for which $\sigma \in \Sigma_\tau$, *silent*. Any input word of the form $u\sigma w$, where $u, w \in \Sigma_\tau^*$ (the Kleene closure of Σ_τ) and $\sigma \in \Sigma_T \setminus \Sigma_\tau$ will be called a *composite transition* of **T** if there are $q_1, \ldots, q_n \in Q$, not necessarily distinct, such that $q_1 \xrightarrow{\sigma_1}_T q_2 \xrightarrow{\sigma_2}_T q_3 \cdots q_{n-1} \xrightarrow{\sigma_{n-1}}_T q_n$ and $\sigma_1 \cdots \sigma_{n-1} = u\,\sigma\,w$. In this case we write $q_1 \overset{\sigma}{\rightsquigarrow} q_n$.

We introduce the notion of a valuation map that will be used to construct the discrete abstraction of the hybrid system 3.1.1.

Definition 3.3.2 (Valuation map)**.** V_M*:* $\mathcal{H} \times \mathcal{P} \mapsto \mathcal{V} \subseteq \{1, 0\}^{|\mathcal{AP}|}$ *is a function that maps pairs* (h, p) *of hybrid states and parameters of* **H***, to binary vectors formed by evaluating each atomic proposition such that whenever* $(h, p) \models \alpha_i$*, we write* $v[i] = 1$ *to denote that the pair* (h, p) *evaluates true for the* i^{th} *predicate* $\alpha_i \in \mathcal{AP}$*.*

We define a relation $\mathfrak{R}$ on $\mathcal{H} \times \mathcal{V}$ according to which we say that a state $h \in \mathcal{H}$ is related to $v \in \mathcal{V}$ if there exists a $p \in s(h)$ such that $(h, p) \models v$, in which case we write $(h, v) \in \mathfrak{R}$.

The discrete abstraction is now defined in a constructive way as follows:

Definition 3.3.3 (Induced transition system)**.** *A hybrid agent* $\mathbf{H}$ *induces a finite labeled transition system* $\mathbf{T}(\mathbf{H})$ *in which:*

1. $\mathcal{Q} \subset \mathcal{V}$ *is the range of* V_M *for all* (h, p) *with* $h \in \mathcal{H}$ *and* $p \in s(h)$*;*
2. $\Sigma_T = \mathcal{K} \cup \{\tau_k : k \in \mathcal{K}\}$*;*
3. $\mathcal{Q}_0 = \{v \in \mathcal{V} \mid \exists h \in \mathcal{H}_0, p \in s(h) : (h, p) \models v\}$*;*
4. $(v, \sigma, v') \in \Delta_T$ *if*
   - $\sigma \in \mathcal{K} \subset \Sigma_T$*, and there exist pairs* (h, p) *satisfying* $V_M(h, p) = v$ *for* $h \in \mathcal{H}$ *and* $p \in s(h)$*, and* (h', p) *satisfying* $V_M(h', p) = v'$ *for* $h' \in s^{-1}(p)$*, such that* $h \overset{\sigma[p]}{\dashrightarrow} h'$*;*
   - $\sigma = \tau_{k'} \in \Sigma_T \setminus \mathcal{K}$*, and there exist pairs* (h, p) *satisfying* $V_M(h, p) = v$ *for* $h \in \mathcal{H}$ *and* $p \in s(h)$*, and* (h, p') *satisfying* $V_M(h, p') = v'$ *for* $p' \in s(h)$*, such that* $(h, p, k) \longrightarrow (h, p', k')$*.*

Note that if h can take a transition, then all h' that can give the same valuation can take the same transition:

Lemma 1. *Suppose that* (h, p) *and* (h', p') *belong in the preimage of* v*, that is* $V_M(h, p) = V_M(h', p') = v$*. If* $(h, p, k) \to (h, p, k')$ *for some* $k, k' \in \mathcal{K}$*, then* $(h', p', k) \to (h', p', k')$*.*

Proof. Writing $V_M(h, p) = v$ implies that a specific combination of atomic propositions are true at state h when the parameter vector is set to p; without loss of generality, let this set of propositions that evaluate true be $\{\alpha_1, \ldots, \alpha_m\}$. From $(h, p, k) \to (h, p, k')$ we conclude that $(h, p) \models \overrightarrow{k}$ and $(h, p) \models \overleftarrow{k'}$. It follows that $\overrightarrow{k} \subseteq \{\alpha_1, \ldots, \alpha_m\} \supseteq \overleftarrow{k'}$. Suppose now that $V_M(h', p') = v$; this means that the same

set of atomic propositions that are true when the system is at state h with parameter p, are also true at state h' with parameter p'. Therefore, we must have $(h', p') \models \overrightarrow{k}$ and $(h', p') \models \overleftarrow{k'}$. According to the definition of T in definition 3.1.1, $(h', p', k) \to (h', p', k') \in T$. □

Consistency between the concrete system and its abstraction is established in terms of a version of weak bisimulation [45].

Definition 3.3.4 (Weak bisimulation of labeled transition systems)**.** *Consider two (labeled) transition systems over the same input alphabet* Σ_T, $\mathbf{T}_1 = (\mathcal{Q}_1, \Sigma_T, \to_1, I_1)$ *and* $\mathbf{T}_2 = (\mathcal{Q}_2, \Sigma_T, \to_2, I_2)$, *and let* $\Sigma_\tau \subset \Sigma_T$ *be a set of input symbols that trigger silent transitions. A binary relation* $\mathfrak{R}$ *on* $\mathcal{Q}_1 \times \mathcal{Q}_2$ *is a* weak bisimulation *if for every* $(q_1, q_2) \in \mathfrak{R}$,

1. $q_1 \overset{\sigma}{\rightsquigarrow}_1 q_1' \quad \Rightarrow \quad \exists\, (q_1', q_2') \in \mathfrak{R} : q_2 \overset{\sigma}{\rightsquigarrow}_2 q_2'$,
2. $q_2 \overset{\sigma}{\rightsquigarrow}_2 q_2' \quad \Rightarrow \quad \exists\, (q_1', q_2') \in \mathfrak{R} : q_1 \overset{\sigma}{\rightsquigarrow}_1 q_1'$.

Then $\mathbf{T}_1$ *and* $\mathbf{T}_2$ *are called weakly bisimilar and we write* $\mathbf{T}_1 \approx \mathbf{T}_2$.

Although this definition relates models of the same type (transition systems), we will abuse it slightly when associating a hybrid system with a transition system. The objects we will relate across systems as well as the transitions considered in this relation, do not necessarily coincide with actual states and transition maps of the underlying dynamical systems; the distinction will be made clear in the following result.

Theorem 3.3.1. *There exists a weak bisimulation* $\mathfrak{R}$ *between a hybrid robotic agent* $\mathbf{H}$ *and its induced finite labeled transition system* $\mathbf{T}(\mathbf{H})$ *and in the sense that:*

1. *If* $(h, v) \in \mathfrak{R} \subset \mathcal{H} \times \mathcal{Q}$ *and* $h \overset{k[p]}{\dashrightarrow} h'$, *then* $\exists\, v' \in \mathcal{Q}$, $v \overset{k}{\rightsquigarrow} v'$ *and* $(h', v') \in \mathfrak{R}$.

2. *If* $(h, v) \in \mathfrak{R} \subset \mathcal{H} \times \mathcal{Q}$ *and* $v \overset{k}{\rightsquigarrow} v'$, *then* $\exists\ h' \in \mathcal{H}$ *such that* $h \overset{k[p]}{\dashrightarrow} h'$ *and* $(h', v') \in \mathcal{R}$.

Then we write $\mathbf{T} \approx \mathbf{H}$.

Proof. To prove the first implication, note that $(h, v) \in \mathfrak{R}$ means by definition that for some p' in $s(h)$, the valuation map at $V_M(h, p') = v$. For generality, assume that $p' \neq p$. Since $h \overset{k[p]}{\dashrightarrow} h'$, p must also be in $s(h)$ and $(h, p) \models \overleftarrow{k}$. Therefore we trivially have $(h, p', k) \longrightarrow (h, p, k)$,[1] and thus by definition 3.3.3 there exists a silent transition $v \overset{\tau_k}{\longrightarrow}_T v''$. The existence of the evolution $h \overset{k[p]}{\dashrightarrow} h'$, also suggests that $h \in s^{-1}(p)$ and $h' \in s^{-1}(p)$. Let $V_M(h', p) = v'$. With $V_M(h, p) = v''$ and $V_M(h', p) = v'$ by definition 3.3.3 there must be a transition $v'' \overset{k}{\longrightarrow}_T v'$. We thus have a composite transition $v \overset{\tau_k}{\longrightarrow}_T v'' \overset{k}{\longrightarrow}_T v'$, which means that $v \overset{k}{\rightsquigarrow} v'$ with $V_M(h', p) = v' \Leftrightarrow (h', v') \in \mathfrak{R}$.

To establish the second implication, observe that if $v \overset{k}{\rightsquigarrow} v'$, then there must be $q, q' \in \mathcal{Q}$ such that $q \overset{k}{\longrightarrow}_T q'$. Given that $(v, h) \in \mathfrak{R}$ and that v jumps to q via a series of silent transitions (in which the hybrid state h is preserved), we can invoke definition 3.3.3 to ensure the existence of an evolution $h \overset{k[p]}{\dashrightarrow} h'$ for some $p \in s(h) \cap s(h')$, such that $V_M(h, p) = q$ and $V_M(h', p) = q'$. It remains to show that $(h', v') \in \mathfrak{R}$. By the same token, q' jumps to v' by another series of silent transitions, in which the hybrid state remains at h'. By definition 3.3.3 therefore, just as we have $p \in s(h')$ and $V_M(h', p) = q'$ there should also be a $p' \in s(h')$ such that $V_M(h', p') = v'$, which shows that $(h', v') \in \mathfrak{R}$. □

What theorem 3.3.1 implies, is that a controller switching sequence given as a succession of T transitions —once interlaced with the corresponding silent transitions that "prepare" the activation of these controllers— yields an acceptable input

[1] These type of transitions in **H** correspond to an on-line re-parameterization of the controller already activated.

trace in the transition system. Conversely, an acceptable input trace in the transition system, once projected on the set of controller symbols (removing the silent transitions) gives a controller sequence for the hybrid system that is implementable. This feature is significant because when one wishes to synthesize motion and control strategies, planning can be performed on the simpler discrete system without worrying about compatibility with the concrete dynamics. In addition, the bisimilarity result ensures that none of the reachability capabilities of the hybrid system on the state space partition created by adding and subtracting PRE and POST sets is lost in the abstraction: whatever transition between these sets the hybrid system can perform, the discrete system can match.

3.4 Application Example: How to Fetch a Printout

In this section we use the abstraction of the mobile robot of figure 1.2(a) to show how a control plan can be devised for the robot to pick up a printout and deliver it to the user who sent it to the printer. In principle, the robot must first go and appropriately "park" in front of the printer location, where the origin of the global coordinate frame is set. Then, it should pick the printout which is at p_o^p, and hand them over to the user at p_o^u.

The robot is assumed modeled in the form of a hybrid robot agent

$$\mathbf{H}_r = \left\{ \mathcal{X}, \mathscr{L}, \mathcal{P}, \mathcal{K}, \mathcal{AP}, f, \overset{\leftarrow}{\cdot}, \overset{\rightarrow}{\cdot}, s, T \right\}$$

with the model components defined as follows.

- $\mathcal{X} = \{q_p, \theta, q_m, q_o\}$ are the continuous states; $(q_p, \theta) \in \mathcal{F}_1 \subset \mathbb{R}^2 \times \mathsf{S}^1$ is the mobile platform's position and orientation, and $\mathcal{F}_1$ represents the obstacle free configuration space. The cartesian position of the manipulator is denoted

$q_m \in \mathcal{W}(q_p) \subset \mathbb{R}^3$, with $\mathcal{W}(q_p)$ being the platform-dependent manipulator workspace; $q_o \in \mathbb{R}^3$ is the position of object in the environment.[2]

- $\mathscr{L} = \{\mathfrak{g}\}$ is a boolean variable $\mathfrak{g} \in \{\mathbf{0}, \mathbf{1}\}$ expressing the state of the gripper. If an object is held $\mathfrak{g} \Leftrightarrow \mathbf{1}$, otherwise $\mathfrak{g} \Leftrightarrow \mathbf{0}$.

- $p = (p_p, p_o) \in \mathcal{P} \subset \mathbb{R}^6$ is the set of system parameters: $p_p \in \mathbb{R}^2 \times \mathsf{S}^1$ is a position and orientation (pose) reference for the platform; $p_o \in \mathbb{R}^3$ is a direction of approach to the position reference for the manipulated object.

- $\mathcal{K} = \{a, b, c\}$ is the set of discrete system modes, marking the particular controller activated on the robot. Symbol a corresponds to the platform's navigation controller that moves the robot from an initial point q_p^i to a desired configuration q_p^d. Symbol b corresponds to the action of picking up an object from location p_o and holding it in a designated "home" position ${q_m^h}''$, the double primes denoting that the home position is soecified with respect to the robot-fixed coordinates. Symbol c corresponds to placing the held object in a specified position at location p_o and returning the arm to ${q_m^h}''$.

- $\mathcal{AP}$ is an indexed set of atomic propositions $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$.

  - $\alpha_1 \Leftrightarrow q_p \in p_p + \mathcal{B}_\varepsilon$, which when true implies that the platform is close to its reference position.[3]

  - $\alpha_2 \Leftrightarrow q_o \in p_o + \mathcal{B}_\varepsilon$, which when true implies that the object is in the neighborhood of location p_o.

[2] Typically, q_o would be part of another hybrid system representing the environment, which is composed with the robotic agent in parallel. For the purposes of this example, however, this variable is lumped together with the robot's states.

[3] $\mathcal{B}_\varepsilon$ is a ball of radius ε that accounts for errors such as robustness, sensing/actuation errors, repeatability errors, etc., inherent in any mechanical system. So long as the position q_p of the robot's base is close enough to p_p upto within the error $\mathcal{B}_\varepsilon$, α_1 evaluates as true.

- $\alpha_3 \Leftrightarrow p_o \in \mathcal{W}(q_p)$, which when true suggests that reference location p_o is within the reachable workspace of the manipulator if its base is located at q_p.
- $\alpha_4 \Leftrightarrow \mathfrak{g}$, is a binary variable indicating the state of the gripper at the robot's end effector.

- f is the vector field given by equation (1.1).
- The PRE and POST conditions for each mode $k \in \mathcal{K}$ are given in table 3.1.
- $s\big((q_p, \theta, q_m, q_o, \mathfrak{g})\big) = \mathcal{F}_1 \times \mathbb{R}^3$.
- T is described generically according to the definition, with the note that there can be no transitions of the form $(h, p, b) \to (h, p', b)$ or $(h, p, c) \to (h, p', c)$, because no matter of the choice of p', the PRE and POST of both b and c are incompatible with each other (α_4 does not depend on parameters). This reflects the fact that once the gripper holds something it cannot pick up something new, and if it has just placed the object somewhere it is not possible to place the same object somewhere else without picking it up first.

Table 3.1: PRE ($\overleftarrow{\cdot}$) and POST ($\overrightarrow{\cdot}$) conditions for the discrete states of the hybrid robotic agent in the example. The sets of atomic propositions are interpreted as conjunctions, *i.e.,* $\{\alpha_2, \alpha_3, \neg\alpha_4\} \Leftrightarrow \alpha_2 \wedge \alpha_3 \wedge (\neg\alpha_4)$, and $\mathbf{1}$ indicates a tautology.

	a	b	c
PRE	$\mathbf{1}$	$\{\alpha_2, \alpha_3, \neg\alpha_4\}$	$\{\neg\alpha_2, \alpha_3, \alpha_4\}$
POST	$\{\alpha_1\}$	$\{\neg\alpha_2, \alpha_3, \alpha_4\}$	$\{\alpha_2, \alpha_3, \neg\alpha_4\}$

Let $\mathcal{H} = \mathcal{X} \times \mathcal{L}$. The transition system induced by $\mathbf{H}_r$ is $\mathbf{T}_r(\mathbf{H}_r) = \{\mathcal{Q}, \Sigma_T, \mathcal{Q}_0, \Delta_T\}$ in which:

- $Q = \{v | \exists h \in \mathcal{H},\, \exists p \in s(h) : V_T(h, p) = v\}$
- $\Sigma_T = \{a, b, c, \tau_a, \tau_b, \tau_c\}$
- $(v, \sigma, v') \in \Delta_T$ if
  - $\sigma \in \{a, b, c\}$, and there exist a pair $(h, p) \in \mathcal{H} \times s(\mathcal{H})$ with $(h, p) \models v$ and a $h' \in s^{-1}(p)$, such that $h \overset{\sigma[p]}{\dashrightarrow} h'$ and $(h', p) \models v'$.
  - $\sigma \equiv \tau_{k'} \in \{\tau_a, \tau_b, \tau_c\}$, and there exist a pair $(h, p) \in \mathcal{H} \times s(\mathcal{H})$ with $(h, p) \models v$ and a $p' \in s(h)$ with $(h, p') \models v'$, such that $(h, p, k) \rightarrow (h, p', k')$.
- $Q_0 = \{v \in Q \mid \exists h_0 \in \mathcal{H}_0,\, \exists p_0 \in s(h_0) : V_T(h_0, p_0) = v\}$.

We proceed to show that a meaningful execution in $\mathbf{H}_r$ that completes the desired task has a complete parallel run on $\mathbf{T}_r$. Let us assume that the object to be manipulated (printout) is at the printer's output tray denoted p_o^r, which marks a vector along which the gripper of the robot's manipulator can grasp the stack of papers. Suppose that the appropriate "parking spot" for the platform that will allow the onboard manipulator to pick up papers from the output tray of the printer is denoted p_p^r. Let the "parking spot" for the platform, next to the desk of the user who ordered the printout be denoted p_p^u, and the location and orientation on the desk where the printout is to be delivered, p_o^u. Let the initial parameter assignment be denoted $p^i = (p_p^i, p_o^i)$. In addition, suppose that initially $\mathfrak{g} \Leftrightarrow \mathbf{0}$ (nothing in the gripper), and that in this initial state h_i, only α_1 evaluates true. Also, the initial value of parameter p_o^i is set outside the workspace of the robot's arm, so that α_3 is false. This implies that $\mathbf{T}_r$ starts at $v_i = 1000$.

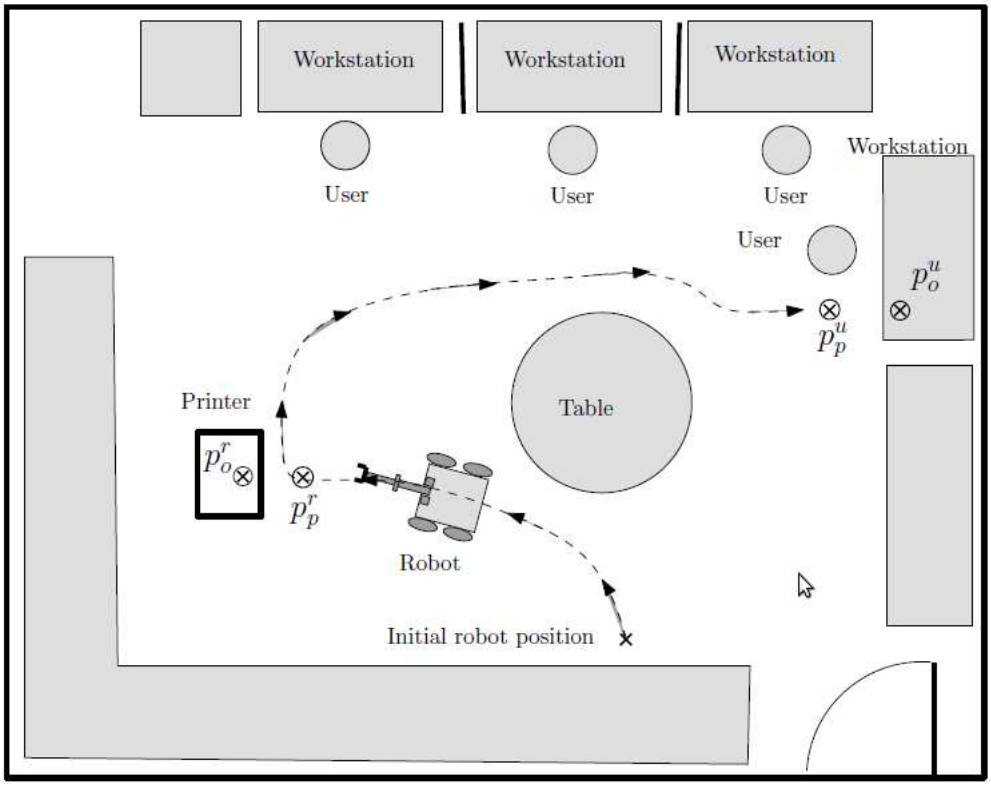


Figure 3.1: The "fetch the printout" scenario, where the the intermediate desired configurations for the mobile platform and the manipulator have been marked.

The reset map can be turned into a mechanism that allows us to control the execution of the hybrid robotic agent. Control design for the agent, in order for it to complete the task, is achieved by refining the reset map as in the following definition:

$$s(h) = \begin{cases} p^1 \triangleq (p_p^r, p_o^i) & \text{if } h = h_i, \\ p^2 \triangleq (p_p^r, p_o^r) & \text{if } h = h_1 \triangleq (p_p^r, q_m^h, p_o^r, \mathbf{0}), \\ p^3 \triangleq (p_p^u, p_o^r) & \text{if } h = h_2 \triangleq (p_p^r, q_m^h, p_p^r + Cq_m^h, \mathbf{1}), \\ p^4 \triangleq (p_p^u, p_o^u) & \text{if } h = h_3 \triangleq (p_p^u, q_m^h, p_p^u + Cq_m^h, \mathbf{1}), \end{cases}$$

where C is the rotation matrix that maps local manipulator coordinates to the inertial coordinate system. This refinement of s essentially tells the robot to first move its platform next the printer at p_p^r, then reach out and pick up the printout at position p_o^r, then navigate to the user desk at p_p^u, and finally leave the printouts at position p_o^u.

To see how this works, note that reseting the parameter vector to $p^1 = (p_p^r, p_o^i)$, results in $(h_i, p_p^r) \models \overleftarrow{a}$, which triggers a transition in $\mathbf{H}_r$ of the form $(h_i, p^i, a) \to (h_i, p^1, a)$. As a result, the hybrid agent moves to discrete location a and the robot starts moving to reposition its platform at location p_p^r where the hybrid state will read

$$h_1 = (p_p^r, q_m^h, q_o^r, \mathbf{0}) \ .$$

Just after the transition occurs in $\mathbf{H}_r$ the valuation map gives $V_M(h_i, p_p^r) = 0000$, and consequently a silent transition occurs in $\mathbf{T}_r$: $1000 \xrightarrow{\tau_a}_T 0000$.

As soon as the robot stabilizes its platform around p_p^r, α_1 becomes true again. System $\mathbf{H}_r$ had its hybrid state evolve from h_i to h_1 due to controller u_a parameterized by p^1:

$$(h_i, a) \to (h_i, a) \overset{a[p^1]}{\dashrightarrow} (h_1, a) \ .$$

During this time the transition system $\mathbf{T}_r$ performs the following jumps:

$$\underbrace{1000}_{V_M(h_i,p^i)} \xrightarrow{\tau_a}_T \underbrace{0000}_{V_M(h_i,p^1)} \xrightarrow{a}_T \underbrace{1000}_{V_M(h_1,p^1)} \ .$$

The reset map triggers a parameter change and activates `pick` controller since its Pre is satisfied. At that point, the reset map dictates that $p := p^2 = (p_p^r, p_o^r)$. Then immediately we have α_2 evaluating true, and since $p_o^r \in \mathcal{W}(p_p^r)$ it will also be that α_3 is true as well. The change of parameters triggers a transition in $\mathbf{H}_r$ from controller u_a to controller u_b, since all the predicates in $\textsc{Pre}(b)$ are true, and parameterized by p^2, controller u_b will move the manipulator to grasp the printout and return the

arm in its home position q_m^h. After the maneuver is completed, the hybrid agent finds itself at state

$$h_2 = (p_p^r, q_m^h, p_p^r + C\, q_m^h, \mathbf{1}) \ ,$$

Then for the hybrid agent we can write

$$(h_1, a) \to (h_1, b) \overset{b[p^2]}{\dashrightarrow} (h_2, b) \ .$$

Then the reset map suggests that the parameter p_o changes from p_o^i to p_o^r. This forces the transition system to first take a silent transition, and then a b-transition:

$$\underbrace{1000}_{V_M(h_1,p^1)} \overset{\tau_b}{\to}_T \underbrace{1110}_{V_M(h_1,p^2)} \overset{b}{\to}_T \underbrace{1011}_{V_M(h_2,p^2)} \ .$$

Further reset of parameter and similar description of transitions in the hybrid and transition system for the `goto` and `place` controllers. Once this is completed, the reset map indicates that $p := p^3 = (p_p^u, p_o^r)$, which triggers a transition in $\mathbf{H}_r$ from b back to a and a subsequent motion of the platform from r to p_p^u:

$$(h_2, b) \to (h_2, a) \overset{a[p^3]}{\dashrightarrow} (h_3, a) \ ,$$

where

$$h_3 = (p_p^u, q_m^h, p_p^u + C\, q_m^h, \mathbf{1}) \ .$$

The transition system now undergoes the following sequence of jumps:

$$\underbrace{1011}_{V_M(h_2,p^2)} \overset{\tau_a}{\to}_T \underbrace{0011}_{V_M(h_2,p^3)} \overset{a}{\to}_T \underbrace{1001}_{V_M(h_3,p^3)} \ .$$

Once at h_3, the reset map changes p_o and sets it to p_o^u, that is $p := p^4 = (p_p^u, p_o^u)$. With this parameter assignment, predicate α_3 becomes true, and now the

conditions in the PRE of c are satisfied triggering a transition in $\mathbf{H}_r$ from a to c. Right after the reset, the manipulator reaches out to location p_o^u and releases the printout there before returning back to its home position. If we denote

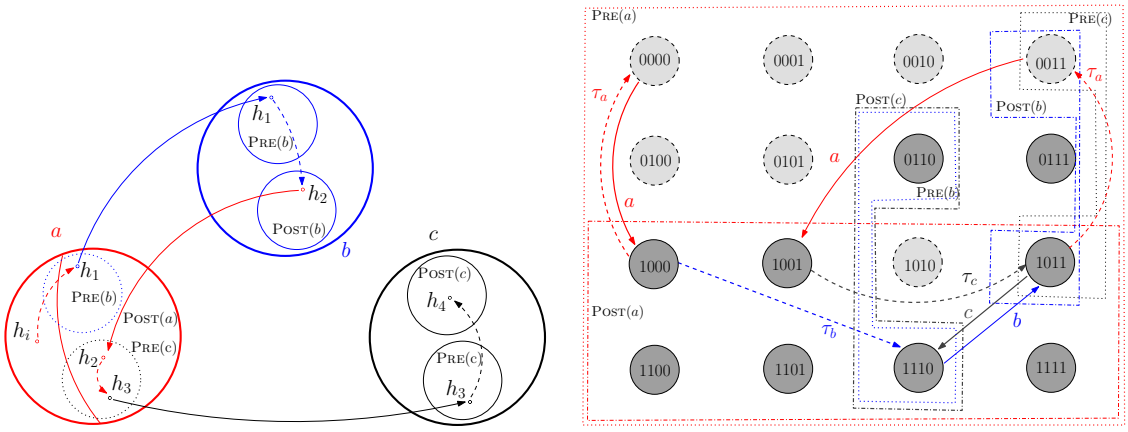
$$h_4 = (p_p^u, q_m^h, p_o^u, \mathbf{0}) \ ,$$

we can express the changes in $\mathbf{H}_r$ as follows:

$$(h_3, a) \rightarrow (h_3, c) \overset{c[p^3]}{\dashrightarrow} (h_4, c) \ .$$

In $\mathbf{T}_r$, on the other hand, we will see the following transitions:

$$\underbrace{1001}_{V_M(h_3,p^3)} \overset{\tau_c}{\rightarrow}_T \underbrace{1011}_{V_M(h_3,p^4)} \overset{c}{\rightarrow}_T \underbrace{1110}_{V_M(h_4,p^4)} \ .$$



(a) Evolution of the hybrid dynamics.

(b) Evolution of the discrete dynamics.

Figure 3.2: The hybrid robotic agent and its abstraction. Continuous evolution and discrete jumps in the hybrid system are mirrored in the silent and regular transitions of the transition system. Each state in the transition system defines a region on the continuous domain where a specific combination of atomic propositions evaluates true.

The execution of the plan to fetch the printout on the hybrid agent $\mathbf{H}_r$ and its abstraction on the transition system $\mathbf{T}_r$ are shown in figure 3.2. To match the transitions of the two systems, one can associate the silent transitions τ_k in the transition system with the jumps between the discrete modes of the hybrid system, and the continuous evolutions in the hybrid system with the regular transitions in the transition system.

3.5 Summary

Stability-based discrete abstractions translate the continuous-time description of a stable dynamical system into a purely discrete form, enabling behavioral modeling using discrete models of computation. This type of abstraction discards the details of state trajectory evolution, preserving only its reachability properties. Based on these ideas, hybrid systems within a certain class can be abstracted into finite state transition systems. Such a finite transition system is shown to be observably bisimilar to the concrete hybrid dynamics it originated from, a fact that ensures that all input strings that the transition system accepts, have a corresponding implementation on the concrete hybrid system. This result allows motion planning and behavior design for the hybrid system to be performed on the discrete system, without concerns about the continuous dynamics of the former.

The transition system exhibits discrete controller transitions executed in a sequence. Such sequences can be considered to be taken from a set that consists of all possible controller sequences executable in the actual hybrid system. Analysis of this set of finite controller sequences leads us to a study of strings of symbols and the sets they form (stringsets, or languages) from a formal language point of view as will be done in the next chapter.

Chapter 4

A SYNERGY BETWEEN ROBOTICS AND LINGUISTICS

In this chapter we study strings of controller sequences accepted by the discrete transition system. The sets of strings called "languages" have been mathematically studied by linguists and computer scientists in formal language theory since the 1950's [24]. Since, the discrete transition system 3.3.1 generates sets of strings, these strings make up a some formal language. This allows us to use the tools developed in formal language theory to study hybrid robotic systems that generate languages by means of control switches.

To start with, we first describe controller sequences that are obtained from the systems that we consider in this work. In the following, we define the term *feasible controller sequence* by considering strings of controller sequences $s = \sigma^{\{1\}}\sigma^{\{2\}}\dots\sigma^{\{n\}}$ accepted by the transition system. These strings do not include silent transitions, but only the observable transitions.

Definition 4.0.1 (Feasible controller sequence)**.** *A controller sequence* $s = \sigma^{\{1\}}\sigma^{\{2\}}\dots\sigma^{\{n\}}$ *where* $\sigma_i \in \Sigma_T/\Sigma_\tau, i = 1, 2, \dots n$*, is called a feasible controller sequence if* $\exists q_0 \in \mathcal{Q}_0, q_n \in \mathcal{Q}$ *and* $\exists \tau_{\sigma^{\{i\}}} \in \Sigma_\tau \cup \varnothing$ *such that for a* $\sigma = \tau_{\sigma^{\{1\}}}\sigma^{\{1\}}\tau_{\sigma^{\{2\}}}\sigma^{\{2\}}\dots \tau_{\sigma^{\{n\}}}\sigma^{\{n\}}$*, we can have* $q_1 \overset{\sigma}{\rightsquigarrow} q_n$*.*

A sequence of controllers $s = k^{\{1\}}k^{\{2\}}\dots k^{\{n\}}$ *is said to be feasible if for each* $k^{\{i\}} \in \mathcal{K}$*, the* Post *of controller* $k^{\{i\}}$ *satisfies the* Pre *of the next controller* $k^{\{i+1\}} \in \mathcal{K}$*. i.e.:*

$$\overrightarrow{k}^{\{i\}}[p^{\{i\}}] = \overleftarrow{k}^{\{i+1\}}[p^{\{i+1\}}] \ \forall i = 1, 2, \dots, n-1.$$

Notice that there can be multiple controllers whose pre-conditions are satisfied after execution of the last controller, provided an appropriate parameter can be found. This leads to non-determinism in the trajectories in $\mathbf{H}$ and allows the robot to "select" an appropriate control sequence according to the task specification. The set of all feasible strings form the *language* of the system. Although, owing to non-determinism, there could be many strings that $\mathbf{T}$ accepts, all the strings belong to the same language that we denote $L(\mathbf{T})$ as the language of the transition system.

Definition 4.0.2 (Alphabet and language of the hybrid robotic system $\Sigma_{\mathbf{T}}$, $L(\mathbf{T})$)**.** *The language $L(\mathbf{T})$ of a transition system $\mathbf{T}$ is the unique set of all feasible controller sequences. The unique set of all controller transitions is said to form the alphabet for the language $\Sigma_{\mathbf{T}} \subseteq \Sigma_T/\Sigma_\tau$.*

For example, the alphabet for the example of section 3.4 is $\Sigma_{\mathbf{T}} = \{a, b, c\}$, that denote `goto`, `pick` and `place` respectively. We will use the notation $\{A, B, C\}$ to mean the same as $\{a, b, c\}$ in the rest of our treatment.

In this chapter, we analyze the languages $L(\mathbf{T})$ to find their place in the Chomsky hierarchy. Such analysis enables us to characterize the complexity of representing and operating on such languages. Furthermore, we also find the position of these languages within the Sub-regular hierarchy to be able to identify specific analysis tools that can be applied on these languages.

4.1 Preliminaries

Let Σ represent the alphabet with Σ^* the Kleene-star of Σ, which is the set of all possible finite strings (including the empty string ϵ) over Σ. The term 'word' and 'sequence' are used interchangeably and is defined as a generic sequence taken from Σ^*. The Kleene-star of Σ is closed under the binary operation of concatenation i.e.: if $s_1, s_2 \in \Sigma^*$, then $s_1 s_2 \in \Sigma^*$. We write $\Sigma^{\leq k}$ to denote the set of all words with length up to k. The length of a word $w \in \Sigma^*$ is denoted by $|w|$, and $|w|_l$ represents

the number of times the string l (which is essentially a controller sequence) appears in the word w. For example, $|baaa|_{ba} = 1$ and $|baaa|_{aa} = 2$. The symbols $\rtimes$ and $\ltimes$ denote the start and end of a string respectively. Given a set A, the notation 2^A means the power set of A, that is, the set of all subsets of A.

Definition 4.1.1 (cf. [8]). *A* deterministic automaton*, denoted by G, is a tuple*

$$G = (Q, \Sigma, \delta, \mathcal{I}, \mathcal{F})$$

where:

1. *Q is the set of states;*
2. *Σ us the finite set of events associated with the transitions in G;*
3. *$\delta : Q \times \Sigma \to Q$ is the transition function: $\delta(x, \sigma) = y$ means that there is a transition labeled by event $\sigma \in \Sigma$ from state $x \in Q$ to state $y \in Q$; in general, δ is a partial function on its domain;*
4. *$\mathcal{I}$ is the initial state;*
5. *$\mathcal{F} \subseteq Q$ is the set of marked states or final states.*

The automaton is said to be *deterministic* because δ is a function over $Q \times \Sigma$. In contrast, the transition structure of a *nondeterministic* automaton is defined by means of a relation over $Q \times \Sigma \times Q$ or, equivalently, a function from $Q \times \Sigma$ to 2^Q [8]. We use the term automaton to refer to a deterministic automaton in this work.

Definition 4.1.2 ([42]). *The following is the definition of a* regular expression *over an alphabet Σ.*

1. *Any letter of the alphabet is a regular expression.*

2. *ϵ (the null word) and $\varnothing$ (the empty set) are regular expressions.*

   If ψ and χ are regular expressions, then all of the following are regular expressions:

3. *$(\psi) \cup (\chi)$ - the union.*

4. *$(\psi) \cap (\chi)$ - the intersection.*

5. *$\sim (\psi)$ - the complement with respect to the set of all words over Σ, with respect to Σ^* i.e., $\sim (\psi) = \Sigma^* - (\psi)$.*

6. *(ψ) (χ) - the concatenation of (ψ) with (χ).*

7. *$(\psi)^*$ - closure, iteration, star closure or Kleene Star.*

4.1.1 The Chomsky Hierarchy

In this section, we define terms from the Chomsky hierarchy. We will only focus on automata theoretic definitions here which is also the focus of our work. Only the definitions that we will be using later for our language analysis have been presented formally, while an informal description of others is presented to familiarize the reader with the Chomsky hierarchy. Other definitions, including formal mathematical descriptions can be found in [24] and [8].

Definition 4.1.3 ([24]). *A* finite state automaton *is a five tuple*

$$A = (Q, \Sigma, \delta, q_0, F).$$

1. *Q is a finite nonempty set of states;*

2. *Σ is the finite nonempty set of inputs;*

3. *δ is a function from $Q \times \Sigma$ into Q called the direct transition function;*

4. $q_0 \in Q$ *is the initial state;*

5. $F \subseteq Q$ *is the set of final states.*

Definition 4.1.4 (Regular languages)**.** *A language is regular if it can be represented by a regular expression. Regular languages are also defined as the class of languages accepted by a finite state automata.*

Definition 4.1.5 (cf. [24])**.** *A* pushdown automaton *is a finite state automaton with an input tape and an auxiliary pushdown stack (or LIFO: last-in, first-out store). A transition from a given state to a new state occurs by reading an input symbol and a symbol from top of the pushdown stack and writing $n \geq 0$ symbols on the top of the stack. When $n = 0$, the top symbol from the stack is erased. Languages accepted by non-deterministic* pushdown automata *are called* context-free languages.

Definition 4.1.6 (cf. [24])**.** *A* Turing machine *is simply a finite-state control device (or a finite state automaton) with a finite but potentially unbounded read-write tape. Languages accepted by* linear bounded Turing machines [1] *are called context-sensitive languages.*

Definition 4.1.7 (cf. [24])**.** *A* recursively enumerable language *is the one for which, for every word (a string) x, there exists a procedure for recognizing the string x, i.e., the procedure should halt after a finite length of time, accepting x, if x is in the language. If x is not in the language, the procedure might halt, rejecting x, or it may never halt.*

4.1.2 The Sub-regular Hierarchy

The Sub-regular hierarchy [56], [54] is a hierarchy of set inclusions of languages within the regular language class of the Chomsky hierarchy. It consists of two branches - the local branch the piecewise branch (figure 4.1).

[1] Turing Machines with a read-write tape bounded by left and right end markers. The machine cannot read or write beyond these markers and stays within bounds.

Definition 4.1.8 (k-factor)**.** *The k factors of a word $w \in \Sigma^*$ are defined as the set of k-length contiguous sub-sequences of a word:*

$$F_k(w) := \begin{cases} v \in \Sigma^k \quad | \exists u, x \in \Sigma^*; w = uvx, & \text{if } |w| \geq k \\ w & \text{otherwise.} \end{cases}$$

For example, the 3-factors of $PQPPQR$ are $\{PQP, QPP, PPQ, PQR\}$.

Definition 4.1.9 (k-Local Grammar)**.** *A k-local grammar is a set of the* permissible *k-factors.*

$$G_{SL_k} \subseteq F_k(\{\rtimes\} \cdot \Sigma^* \cdot \{\ltimes\})$$

For example, a $2-$local grammar is $G_{SL_2} = \{\rtimes a, aa, ab, ba, bb, b\ltimes\}$. Note that the 2-factors not in this list $\{\rtimes b, a\ltimes\}$ are the *forbidden* factors.

Definition 4.1.10 (cf. [54])**.** *A stringset L over Σ is strictly k-local iff there is some strictly k-local grammar G over Σ (for some k) such that L is the set of all strings that satisfy G. Hence the all the words in a strictly k-local language can be generated by the corresponding k-local grammar.*

$$L(G_{SL_k}) := \{w \in \Sigma^* \mid F_k(\rtimes \cdot w \cdot \ltimes) \subseteq G_{SL_k}\}.$$

Continuing the example of section 3.4, the language of G_{SL_2} is all words which do not contain the forbidden factors. As a regular expression, this language is $a\Sigma^*b$. The set of all strictly k-local languages forms the language class denoted $\mathcal{L}_{SL_k}$. In other words,

$$\mathcal{L}_{SL_k} = \{L(G) \mid G_{SL_k} \subseteq F_k(\{\rtimes\} \cdot \Sigma^* \cdot \{\ltimes\}\}.$$

Definition 4.1.11 (cf. [54]). *A language is said to be locally k-testable (LT_k) iff any two words in the language have exactly the same set of k-factors. The formal definition is below:*

$$L \in LT_k \text{ iff } \forall w, v \in \Sigma^*, \text{ whenever}$$
$$F_k(\rtimes \cdot w \cdot \ltimes) = F_k(\rtimes \cdot v \cdot \ltimes)$$
$$\text{then either } v, w \in L \text{ or } v, w \notin L.$$

While SL_k languages can forbid some sequences from occuring in a language, they cannot require some sequence to occur. This is one example of the additional expressivity of the LT_k class has over the SL_k class.

Definition 4.1.12 (cf. [54]). *A language is said to be locally threshold k-testable up to t ($LTT_{k,t}$) iff deciding whether a word belongs to the language depends only on its* multiset *of k-factors up to some maximum counting threshold t. In other words, languages may distinguish words with the same set of k-factors, provided they have different number of occurances of these factors (up to some threshold t).*

$$\exists k, t \text{ such that } \forall w, v \in \Sigma^*, \text{ if}$$
$$\forall l \in F_k(\rtimes \cdot w \cdot \ltimes) \cup F_k(\rtimes \cdot v \cdot \ltimes)$$
$$\text{either } |w|_l = |v|_l \text{ or both } |w|_l \geq t \text{ and } |v|_l \geq t$$
$$\text{then } w \in L \Leftrightarrow v \in L.$$

The proper inclusion relationships between these three classes (SL, LT, LTT) are summarized in Figure 4.1 [42].

While k-factors capture adjacency relationships, *subsequences* capture long-distance relationships between symbols.

Definition 4.1.13 (cf. [55]). *A subsequence of a word is defined as permutations of the strings formed from the symbols of the word while retaining their relative*

ordering. Hence a subsequence w of string v is a partial order over Σ^* defined as:

$$w \sqsubseteq v \text{ iff } w = \sigma_1\sigma_2 \dots \sigma_n \text{ and } v \in \Sigma^*\sigma_1\Sigma^*\sigma_2\Sigma^* \dots \Sigma^*\sigma_n$$

where $\sigma_i \in \Sigma$.

For example, if $v = PQPPQR$, then $P \sqsubseteq v$, $QR \sqsubseteq v$, $PPR \sqsubseteq v$ and so on.

Definition 4.1.14 (cf. [55])**.** *For $w \in \Sigma^*$, we define the subsequences of length k of a word w as the k-subsequences $P_k(w)$.*

$$P_k(w) := \{v \in \Sigma^k | v \sqsubseteq w\}$$

Similarly we can also define subsequencs of length up to k as:

$$P_{\leq k}(w) := \{v \in \Sigma^{\leq k} | v \sqsubseteq w\}$$

For example, $P_3(PQPPQR) = \{PQP, PQQ, PPP, PQR, PPR, QQR, QPQ, QPR, QPP\}$.

Definition 4.1.15 (cf. [55])**.** *A string w belongs in a Strictly k-Piecewise (SP_k) Language if its k-subsequences are a subset of the permissible k-subsequences that define the grammar of the SP_k language.*

$$\begin{aligned} &G_{SP_k} \subseteq P_{\leq k}(\Sigma^*) \\ &L(G_{SP_k}) = \{w \in \Sigma^* \mid P_{\leq k}(w) \subseteq G_{SP_k}\} \ . \\ &\mathcal{L}_{SP_k} = \{L(G) \mid G \subseteq P_{\leq k}(\Sigma^*)\} \end{aligned}$$

For example, let $\Sigma = \{a, b, c\}$ and $G_{SP_2} = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, cb, cc\}$. G_{SP_2} is the set of permissible subsequences up to length 2. Note that the only forbidden sequence is *ca*. Then the language of this grammar is all words which do not contain this forbidden subsequence (i.e a is not permitted after c so *bbcbbbbbbbbbabbb*

does not belong to this language). As a regular expression, we can write this language as $\sim (\Sigma^* c \Sigma^* a \Sigma^*)$.

Simon [56] introduced the Piecewise Testable languages.

Definition 4.1.16 (cf. [55]). *A word w belongs in a Piecewise k-Testable (PT_k) language iff deciding whether it belongs to the language depends only on its subsequences up to length k $P_{\leq k}(w)$.*

$$L \in PT_k \text{ iff } \forall w, v \in \Sigma^*, \text{ whenever}$$
$$P_{\leq k}(w) = P_{\leq k}(v)$$
$$\text{then either } v, w \in L \text{ or } v, w \notin L.$$

The class of PT_k languages properly includes the strictly k-piecewise languages (figure 4.1).

Definition 4.1.17 (cf. [42]). *A language is star free (SF) if it can be represented by a star free regular expression, a regular expression that can be written without the use of the Kleene-Star "$*$".* [2] *The mathematical definition for the star free class is as follows:*

$$L \in SF \text{ if } \exists n \text{ such that } \forall u, v, w \in \Sigma^*$$
$$\text{if } uv^n w \in L, \text{ then } uv^{n+1} w \in L.$$

Star free languages are a super-class of the local and piecewise branches of the Sub-regular hierarchy (figure 4.1). In other words, the languages that belong to either of the $PTT_{k,t}$ or the LT_k classes also belong to the star free class, though the converse is not true.

[2] The class of star free languages is sometimes called the *non-counting languages*. There are several equivalent definitions of this class, including automata theoretic and algebraic ones [42].

4.1.3 Regular Robotic Languages

Constraints in controller sequencing naturally arise due to PRE-POST compatibility. In general, the following constraints, each defining a different language are seen to be generated from the hybrid systems that we consider.

1. *Adjacency constraints:* The PRE and POST conditions of controllers impose *adjacency constraints* between controllers. Such adjacency relationships naturally gives rise to an SL_2 grammar of feasible controller pairs. For example, for a robot capable of navigating (controller A), picking up objects (B) in its workspace and placing (C) them elsewhere, AB, BA, AA and BC belong to the SL_2 grammar whereas pairs like BB, CC do not. This happens due to the inability of the robot's end effector or gripper to grasp another object while one picking operation has just been completed or place an object just after a preceding place control action. However, we can see that with AB, BA and BC within the SL_2 grammar, the sequence BAAAB belong to the SL_2 language whereas it does not form a feasible controller sequence since there must be a C controller that "frees" the gripper before another B can be executed. We denote the class of SL_2 languages generated by SL_2 grammars of controller pairs as $\mathcal{L}_{SL_2}$.

2. *Long Distance (LD) Constraints:* In addition to the adjacency relationships there could be *long distance dependencies* between two or more controllers. This happens due to inability of a controller to change the value of enough predicates so that the PRE of the other controller can be satisfied. The example 3.4 exemplifies this constraint with a long distance constraint between B and C which says that there must be a C between adjacent Bs and vice-versa. These long distance dependencies can be captured using what we call the *LD-based SL_2 grammar* G_{LD} generating the languages $\mathcal{L}_{LD}$. The SL_2 behavior arises in our case since we only consider the long distance constraints between pairs

of controllers. There could be more than two controllers that exhibit coupled LD behavior, but we keep such SL_k based LD languages for arbitrary k as future work.

To define the $\mathcal{L}_{LD}$ we first introduce a *projection function* E_{LD} acting on a string w as the (sub-)string containing only the symbols/controllers that are LD dependent. E_{LD} ignores all other symbols from the string w other than the LD constrained symbols that "we care about." Let $\mathcal{T}$ be this set of controllers that exhibit LD dependencies and define the grammar $G_{LD} \subseteq \mathcal{T} \times \mathcal{T}$. Then, with $w = \sigma_1\sigma_2 \dots \sigma_n$,

$$E_{LD}(w) = v_1 v_2 \dots v_n \text{ where } \begin{array}{ll} v_i = \sigma_i \text{ iff } \sigma_i \in \mathcal{T} \\ \epsilon \text{ otherwise.} \end{array} \tag{4.1}$$

The empty string ϵ serves to "erase" the controllers that are not long distance dependent. The language of a grammar is defined as

$$L(G_{LD}) = \{w \in \Sigma^* \mid F_2(E_{LD}(w)) \subseteq G_{LD}\}. \tag{4.2}$$

The LD-based SL_2 languages $\mathcal{L}_{LD}$ are defined as the languages generated from the LD constraints.

$$\mathcal{L}_{LD,\mathcal{T}} = \{L(G_{LD}) \mid G_{LD} \subseteq \mathcal{T}^2\}.$$

It is to be noted that $\mathcal{L}_{LD,\mathcal{T}}$ are not SL_2 languages, but LD-based SL_2 languages which means that the substrings of LD dependent controllers $\mathcal{T}$ of the input string follow an SL_2 grammar. The strings satisfying G_{LD} are of the form $U_1^* v_1 U_2^* v_2 U_3^* v_3 \dots v_n U_n^*$ with $v_i \in \mathcal{T}$. It is then seen that $G_{LD} \subseteq GSL_2$ when we observe that any of the $U_i^* \in (\Sigma - \mathcal{T})^*$ can be the empty symbol ϵ. We use the

notations $\mathcal{L}_{LD,\mathcal{T}}$ and $\mathcal{L}_{LD}$ interchangeably to denote the LD-based SL_2 languages.

The class of hybrid systems we consider only exhibits these two constraints in general. Thus the languages generated by an intersection of these two constraints form a *regular robotic language*.

Definition 4.1.18 (Regular robotic languages (RRLs))**.** *Regular robotic languages are the class of languages generated by the finite transition system of definition 3.3.1. They are defined as any finite intersection of languages from* $\mathcal{L}_{SL_2} \cup \mathcal{L}_{LD,\mathcal{T}}$.

Example 1. *Referring back to the example in section 3.4, the* Pre-Post *compatibility between two controllers naturally define an* SL_2 *grammar that can be derived by looking at table 3.1. The language for the hybrid system can be generated by considering that the controllers follow the following constraints.*

- *Starting string constraint:*
  For example, strings beginning with A^*C *are not in the language since to activate the C, we need* α_4 *to evaluate as true, and that can only be affected using a B controller.*

- SL_2 *pattern based on* Pre *and* Post *conditions: For example, controllers A and B can be concatenated iff the* Post *of A satisfies the* Pre *of B. Similar argument holds for other combinations of controllers.*

- *Long distance dependency constraints:*
  For example, there must be a C between two Bs: again a restriction due to the α_4 *predicate since the robot cannot pick two objects at the same time.*

The Pre-Post *based* SL_2 *grammar for our example is the set* $(\rtimes\ltimes, \rtimes A, \rtimes B, AA, AB, BC, AC, CA, CB, C\ltimes, B\ltimes, A\ltimes)$. *However, the language of the robot is not derived from this grammar since otherwise invalid strings like* $BABACAC$ *or*

CAAB would belong to the language. The first one violates the long-distance dependency constraints while the latter violates the starting string constraint.

From these constraints, the regular robotic language for our robot can be written in the form of a regular expression

$$L = A^*(BA^*CA^*)^*(\epsilon \cup B)A^* \tag{4.3}$$

which denotes a robot capable ot picking and placing objects in its workspace any number of times. Also, by definition of regular languages, the language considered here is indeed expressible as a regular expression and is hence a regular language.

In section 4.2, we show that regular robotic languages are star free languages. It is also shown in section 4.2 that the robotic languages are neither locally threshold testable nor piecewise testable.

4.2 Position of RRLs Within the Sub-regular and the Chomsky Hierarchies

Since the robotic languages can be expressed by finite state automata (FSA), they fall under the regular class of languages in the Chomsky hierarchy. Here we are interested in exploring the Sub-regular class of languages. We first show that regular robotic languages are star free languages. We then show by counter-examples that words in a regular robotic pattern belong neither to the local nor the piecewise branches of the Sub-regular hierarchy (figure 4.1).

Theorem 4.2.1. *The regular robotic languages belong to the star free class of the Sub-regular hierarchy.*

Proof. The robotic language we have is

$$RRL = \{L_1 \cap L_2 \cap L_3 \cdots \cap L_n | L_i \in \mathcal{L}_{SL_2} \cup \mathcal{L}_{LD,\mathcal{T}} \text{ for } i = 1, 2, \ldots, n\} \tag{4.4}$$

Since the star free class is closed under intersection [42] and it is known that $\mathcal{L}_{SL_2}$ is a sub-class of star free it only remains to be shown that $\mathcal{L}_{LD,\mathcal{T}}$ is star free. An intersection of these languages will then also be star free by the closure under intersection property.

Consider any language L in $\mathcal{L}_{LD,\mathcal{T}}$. By definition there is some set of long distance dependent controllers $\mathcal{T}$ and $G_{LD} \subseteq \mathcal{T}^2$ such that $L = L(G_{LD})$. It follows that any word w in L is of the form

$$w = (\Sigma - \mathcal{T})^* v_1 (\Sigma - \mathcal{T})^* v_2 \dots v_n (\Sigma - \mathcal{T})^*$$

where $v_i \in \mathcal{T}$. But $(\Sigma - \mathcal{T})^* = \sim (\Sigma^* \mathcal{T} \Sigma^*)$ and $\Sigma^* = \sim \varnothing$ since $\varnothing$ is the empty set. Using these two relationships, we can write w as:

$$w = \sim (\sim \varnothing \mathcal{T} \sim \varnothing) v_1 \sim (\sim \varnothing \mathcal{T} \sim \varnothing) v_2 \dots v_n \sim (\sim \varnothing \mathcal{T} \sim \varnothing)$$

which is a regular expression without the Kleene-star. Hence, the languages $\mathcal{L}_{LD,\mathcal{T}}$ are star free. □

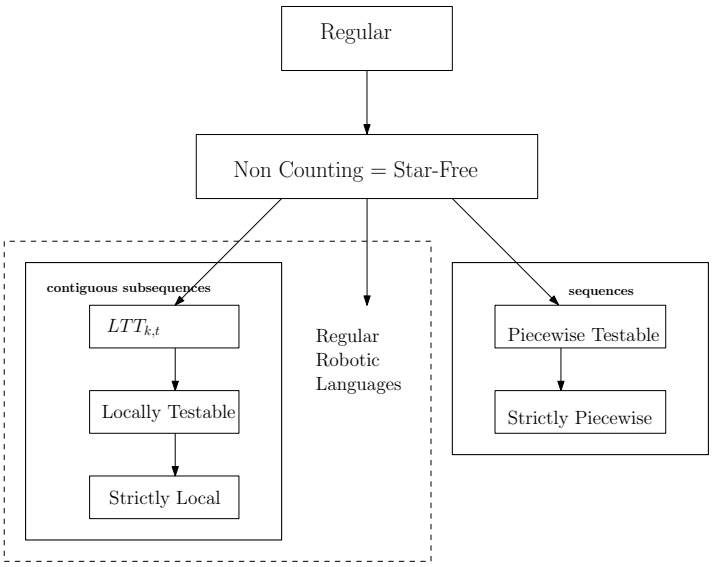


Figure 4.1: Location of regular robotic languages within the Sub-regular hierarchy.

Theorem 4.2.2. *The robotic patterns do not belong to the local branch of the Sub-regular hierarchy.*

Proof. It is sufficient to prove this theorem with a counter-example. Referring to the definition of the locally threshold testable class, we observe that $\forall k$ and any t, consider $w = A^kBA^kBA^kCA^k \notin L$ and $v = A^kBA^kCA^kBA^k \in L$. Then, we have $F_k(\rtimes \cdot w \cdot \ltimes) = F_k(\rtimes \cdot v \cdot \ltimes)$ and $\forall l \in F_k(\rtimes \cdot w \cdot \ltimes)$ it is the case that $|w|_l = |v|_l$

To better understand this argument, take for example $l = A^{k-1}B$ which is a $k-$factor of both v and w, and we see that $|w|_{A^{k-1}B} = |v|_{A^{k-1}B} = 2$. Therefore, two words v and w have the same k-factors and the same number of them, but do not both belong to the example robotic language. Therefore, by definition of the LTT class, this language is not in LTT. Also, it is to be noted that since the the LT and the SL classes are sub-classes of the LTT class, we have actually proved that the robotic languages do not belong to any of these classes and hence, do not belong to the local branch fo the Sub-regular hierarchy. □

Theorem 4.2.3. *The regular robotic languages do not belong to the piecewise branch of the Sub-regular hierarchy.*

Proof. Again, to prove this theorem, we use a counter-example and it suffices to show that the languages do not belong to the piecewise testable class for any k, since the strictly piecewise class is a subclass of piecewise testable class (see figure 4.1). Consider $w = A^k(BA^kBA^kCA^kCA^k)^k \notin L$ and $v = A^k(BA^kCA^kBA^kCA^k)^k \in L$. But, we can see that $P_{\leq k}(w) = P_{\leq k}(v)$. Hence, even though the two words have exactly the same k-subsequences (for any k), not both the words are in the language and are therefore not piecewise testable by definition. □

4.3 Autosegmental Patterns

In this section, we aim to show some parallels between a branch of phonology called the *autosegmental phonology*. We present a rather informal description of autosegmental phonology here and show how such a theory can be appropriately applied to regular robotic languages. The work in this section can be treated as an inspiration to carry out a more formal future work in this area.

Autosegmental phonology ([21]) treats phonological representations as multi-dimensional, having several tiers. This study of phonology, essentially breaks up strings of phonological patterns into (autonomous) phonological tiers according to some property of interest. For example, one might be interested in studying the vowel sounds in a word, without regards to the consonents, so that the phonological tier of interest in the Finnish word *päivää* 'hello', simply the vowels in order without the consonants: *äiää* [26]. There are two tiers in this example that can be represented as shown in figure 4.3. The tiers are connected with association lines to show the relationship between them. In the case of figure 4.3, this relationship shows the position of the vowels from which the upper tier is formed.

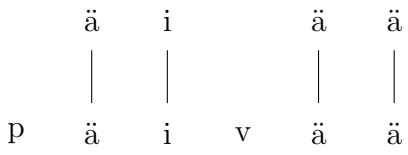


Figure 4.2: Autosegmental tiers of the finnish word *päivää*.

While phonologists use autosegments to describe phonological language rules, we can exploit the concept to describe long distance and other constraints arising due to interacting controllers. The local and long-distance constraints found in regular robotic languages form *autonomous* languages, i.e., each constraint defines a language that can be constructed independently, without a knowledge of the language constructed by the other constraints. The contribution of each of the languages formed by the constraints, can be shown on tiers, which simplify the analysis of the language. With such a logical decomposition of the language into tiers, we see that a simple cross product of the automata, accepting the language on each tier, gives us the robotic language. Such representation allows us to look at the individual languages and their intersection on a single representation.

We now move on to see how regular robotic languages can actually be captured by autosegmental patterns.

Definition 4.3.1. *We define an* SL_2 base tier *as the set of all controllers that exhibit adjacency constraints (see section 4.1.3) for an* SL_2 *language. In other words, the set of controllers that define a language from the class* $\mathcal{L}_{SL_2}$*, form the* SL_2 *base tier.*

There are as many base tiers, as the the SL_2 languages forming the regular robotic language (4.4).

Definition 4.3.2. *An LD based* SL_2 *tier* $\mathcal{T}$ *is defined as the set of controllers that exhibit long-distance dependencies, which describe a language from the class* $\mathcal{L}_{LD,\mathcal{T}}$ *defined in equation* (4.2).

We have already seen that an intersection of languages in $\mathcal{L}_{SL_2} \cup \mathcal{L}_{LD,\mathcal{T}}$ gives us regular robotic languages (4.4). This implies that intersection of all the languages on the tiers in the autosegmental domain gives regular robotic languages.

Figure 4.3 shows some example strings and how the the idea of the tiers for our fetch-a-printout example enable us to identify invalid strings in the language. The LD based tier consists of the symbols B and C. The first two strings in the figure violate the starting string and long distance constraints (see example 1) respectively. Although the examples in figure 4.3 show only two tiers, it is possible to have multiple tiers with each of them addressing a different set(s) of constraints. Violation of constraints in any of the tiers puts the string out of the particular language.

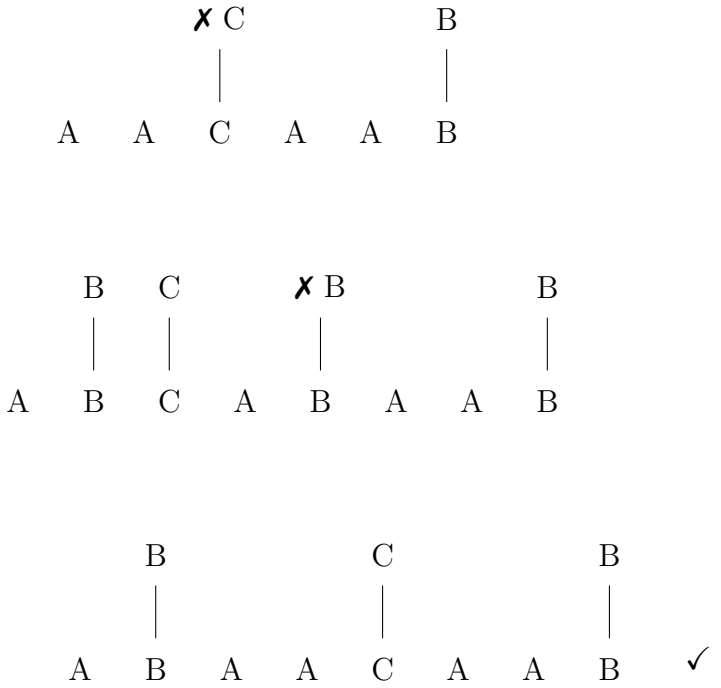


Figure 4.3: Figure describing how autosegments enable a clear understanding of acceptable words.

4.4 Automata Representation of Regular Robotic Languages

This section serves to translate the regular expressions and the languages for the constraints discussed in section 4.1.3 to an automata theoretic framework.

4.4.1 Automata for the SL_2 Languages

A Strictly 2-Local language naturally arises due to interacting $\overleftarrow{\cdot}$ and $\overrightarrow{\cdot}$ of the hybrid system $\mathbf{H}$. Hence, the set of propositions for the robot example given in table 3.1 are a basis to construct an SL_2 grammar. For example, the SL_2 grammar for our fetch-a-printout example can be derived from table 3.1:

$$\mathcal{G} = \{\rtimes A, \rtimes B, AA, AB, BA, AC, CA, CB, BC, C\ltimes, B\ltimes, A\ltimes, \rtimes\ltimes\}. \tag{4.5}$$

We can define an automaton that would accept the language of the SL_2 grammars generated from the adjacency constraints as:

Definition 4.4.1 (SL_2 Automaton)**.** *An automaton accepting an SL_2 language generated by the* Pre-Post *conditions can be defined as a five tuple $\mathbf{A}_B = (\mathcal{Q}_B, \Sigma_B, \mathcal{I}_B, \mathcal{F}_B, \delta_B)$.*

- $\Sigma_B = \Sigma_T/\Sigma_\tau$ *is the set of controllers that define transitions between the states of the automaton.*
- $\mathcal{Q}_B = \{q \Leftrightarrow \exists y \in \Sigma_B \cup \{\rtimes\}$ *such that* $yq \in \mathcal{G}\ \}$ *are the set of states. We mark the states according to the last transition that is taken in the automaton. Here, we use the same symbol for the states and the controllers so that if the last transition taken in the automaton is B, then the system reaches state B.*
- $\mathcal{I}_B \subseteq \mathcal{Q}_B$ *are the initial states. For our transition systems, these are the states that allow transitions that can satisfy the $\overleftarrow{\cdot}$ of any controller with only a change of parameter. In other words, all the states that allow the execution of a controller by only taking a silent transition (and not a concrete transition) are the initial states.*

- $\mathcal{F}_B = \mathcal{Q}_B$ *are the final states. Assuming the robotic system can stop after execution of any controller, we have all states as final states.*

- $\delta_B(q_1, q_2) = q_2 \Leftrightarrow \overrightarrow{q_1}|_{p_1} = \overleftarrow{q_2}|_{p_2}$ *for some* $p_1, p_2 \in \mathcal{P}$ *is the transition function that is based on the* Pre-Post *compatibility between two controllers.*

For the grammar of equation (4.5), this definition gives us the automaton shown in figure 4.4.

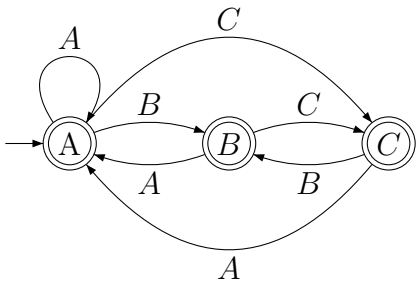


Figure 4.4: SL_2 base tier automaton for the grammar (4.5)

It can be seen from figure 4.4 that all A transitions get the machine only to state A. In other words, the machine keeps track of the last transition seen. This is a property unique to automata that accept SL_2 languages.

4.4.2 Automata for the LD-based SL_k Languages

The LD-based SL_k tier is basically a projection of the the base language that contains only the symbols that exhibit long-distance dependencies. The LD-based SL_k tier represents LD-based SL_k languages $\mathcal{L}_{LD,\mathcal{T}}$ (equation (4.2)). The automata that accept these languages are defined below.

Definition 4.4.2 (LD based strictly k-local automata)**.** *For a tier*

$$\mathcal{T} \subseteq \Sigma_B,$$

the LD based strictly k-local automata can be defined as a five tuple

$$\mathbf{A}_T = (\mathcal{Q}_T, \Sigma_T, \mathcal{I}_T, \mathcal{F}_T, \delta_T)\,,$$

where

- $\mathcal{Q}_T = \mathcal{T} \cup \{\rtimes\}$ *are the set of states that are marked by the symbols in the tier and a start symbol* $\rtimes$*.*
- $\mathcal{I} = \{\rtimes\}$ *is the start state.*
- $\mathcal{F}_T = \mathcal{Q}_T$ *are the final states, assuming all states are final.*
- $\delta_T(q, a) = a \Leftrightarrow a \in T$ *and* $qa \in \mathcal{G}$
  $\delta_T(q, a) = q \Leftrightarrow a \notin T$ *is the transition function that only allows a transition to another state marked by the last controller run on the system, if the controller being run belongs to the tier. Otherwise, the machine takes a self-loop transition to the same state (for controllers that do not belong in the tier).*

The difference between the SL_2 automata and the LD based SL_k automata should be noted. While in the former, the states can be determined by looking at only the last transition of the system, the states of the latter are determined by the last transitions that belong to the tier. Transitions not in the LD tier do not change the state of the LD based SL_k automata.

Figure 4.5 shows the automaton for the example of section 3.4. The LD-based grammar for this example is:

$$G_{LD} = \{\rtimes B, BC, CB, C\ltimes, B\ltimes, \rtimes\ltimes\} \subseteq \mathcal{G}_B \tag{4.6}$$

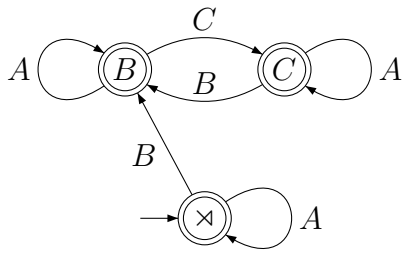


Figure 4.5: Tier based automaton.

At this point, we have all the ingredients to obtain the robotic type patterns we have been seeking. The languages are described by the product automaton described below.

4.4.3 The Product Automaton

The regular robotic languages are an intersection of SL_2 and tier based SL_2 patterns as shown in equation (4.4):

$$RRL = \{L_1 \cap L_2 \cap L_3 \cdots \cap L_n | L_i \in \mathcal{L}_{SL_2} \cup \mathcal{L}_{LD,\mathcal{T}} \text{ for } i = 1, 2, \ldots, n\}$$

Since the product of automata generating these patterns gives us a language that is the intersection of these patterns ([8, page 84]), the product of the SL_2 automata and the tier-based SL_2 automata gives us the product automaton that describes regular robotic languages.

Here we define the product of two automata $G_1 = (Q_1, \Sigma_1, \delta_1, \Gamma_1, \mathcal{I}_1, \mathcal{F}_1)$ and $G_1 = (Q_2, \Sigma_2, \delta_2, \Gamma_2, \mathcal{I}_2, \mathcal{F}_2)$. The languages accepted by each automaton is denoted $L(G_1)$ and $L(G_2)$ respectively. The definition for the product can be generalized for more than two automata following the same syntax as for the product of two automata.

Definition 4.4.3 (cf. [8]). *The product of two automata G_1 and G_2 is defined as:*

$$G_1 \times G_2 \ = (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, \mathcal{I}_1 \times \mathcal{I}_2, \mathcal{F}_1 \times \mathcal{F}_2) \tag{4.7}$$

where

$$f((q_1, q_2), \sigma) := \left\{ \begin{array}{ll} (f_1(q_1, \sigma), f_2(q_2, \sigma)) & \textit{if } \sigma \in \Gamma_1(q_1) \cap \Gamma_2(q_2) \\ \textit{undefined} & \textit{otherwise.} \end{array} \right\}. \tag{4.8}$$

For the example of section 3.4, the automaton that describes the regular robotic language (4.3) is a product of the two automata in figures 4.4 and 4.5. The product automaton for this example is shown in figure 4.6.

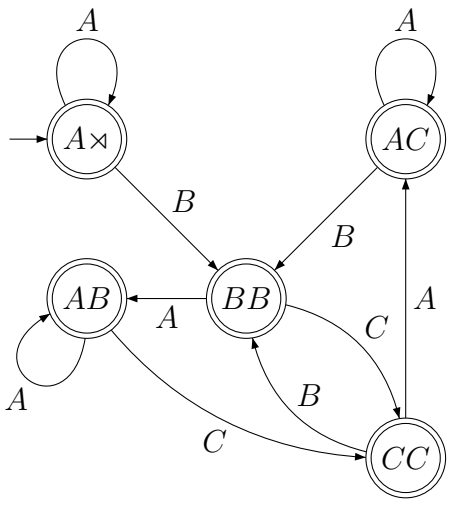


Figure 4.6: The product automaton.

It can be seen that the language accepted by this automaton is

$$L = A^*(BA^*CA^*)^*(\epsilon \cup B)A^*$$

as intended.

4.5 Summary

Representing strings of controller sequences in terms of languages offers the benefit of applying computational analysis and formal language theory to the systems that generate these controller sequences. We have shown that regular robotic languages can be expressed as regular expressions, which corroborates the fact that the languages are indeed regular. Regular languages can be accepted by finite state automata without the use of any additional memory, as is required by automata accepting richer classes of languages in the Chomsky hierarchy. This requires less computational effort when performing operations on regular robotic languages.

Furthermore, in our quest to find the smallest known class to which regular robotics languages belong, we have been able to prove that this class is the star free languages. Star free languages offer many useful properties such as closure under intersection, union and concatenation that can be exploited to study interactions with other robotic languages, thereby opening doors to a study of multi-robot planning using a linguistics framework. Additionally, star free expressions can be represented by a wide number of equivalent tools including automata, logic formulas, etc., as described in [42], which widens the horizons for study of hybrid robotic systems with asymptotically stable continuous dynamics.

Lastly, we have shown that our languages can be represented in terms of autosegmental tiers, similar to those seen in autosegmental phonology. Although we did not explore this area in detail, the parallels between linguistics and robotics that we have drawn in this area offer an additional tool to exploit for the analysis

of robotic systems that generate regular robotic languages. In the least, we have been able to decompose the star free robotic language into SL_k based ones that offer immediate benefits in terms of automata operations. For example, SL_k languages are learnable by a special class of language learners called *string extension learners* [25].

Chapter 5

CASE STUDY

In this chapter we apply the ideas developed so far to control an actual robot, modeled as a hybrid system. We express the ideas through a physical realization of the example introduced in section 1.1 of chapter 1, and explained in section 3.4 of chapter 3. We use the same notation as used in the previous chapters. Therefore, we would recommend that the reader refers to the sections mentioned above before proceeding with this case study.

5.1 Coordinate Frames and Equipment

We use three cartesian coordinate frames, that will described shortly, for this case study. The two major hardware components that we use in our experiments are a physical robot and a motion capture system that provides feedback on cartesian coordinates of the robot and elements in the workspace environment. Each of these are briefly introduced in this section.

5.1.1 Coordinate Frames

We use three orthogonal cartesian coordinate frames $CF_{global}, CF_{rotated}$ and CF_{local}, to express different quantities as we proceed in this case study. Unit vectors $(\mathbf{i}, \mathbf{j}, \mathbf{k}), (\mathbf{i}', \mathbf{j}', \mathbf{k}')$ and $(\mathbf{i}'', \mathbf{j}'', \mathbf{k}'')$ are assumed along the three orthogonal directions of coordinates $CF_{global}, CF_{rotated}$ and CF_{local} respectively (see figure 5.1). Also we express a, a' and a'' to denote the representation of a, with respect to the frames $CF_{global}, CF_{rotated}$ and CF_{local} respectively.

Refering to figure 5.1, the global coordinate frame CF_{global} is fixed at a point in the workspace where we perform the experiments, with the **k** direction pointing perpendicular outward from the ground, and the **i** and **j** directions arbitrarily fixed at the origin O_g. The origin O_g is an arbitrarily chosen point in the workspace. The robot-fixed coordinate frame CF_{local} has its origin O_l fixed at the point where the arm connects with the base on the robot. The $\mathbf{j}''$ direction of frame CF_{local} is parallel to the **k** direction of CF_{global}, while the $\mathbf{i}''$ direction in CF_{local} extends parallel to the robots base towards the forward direction of motion.

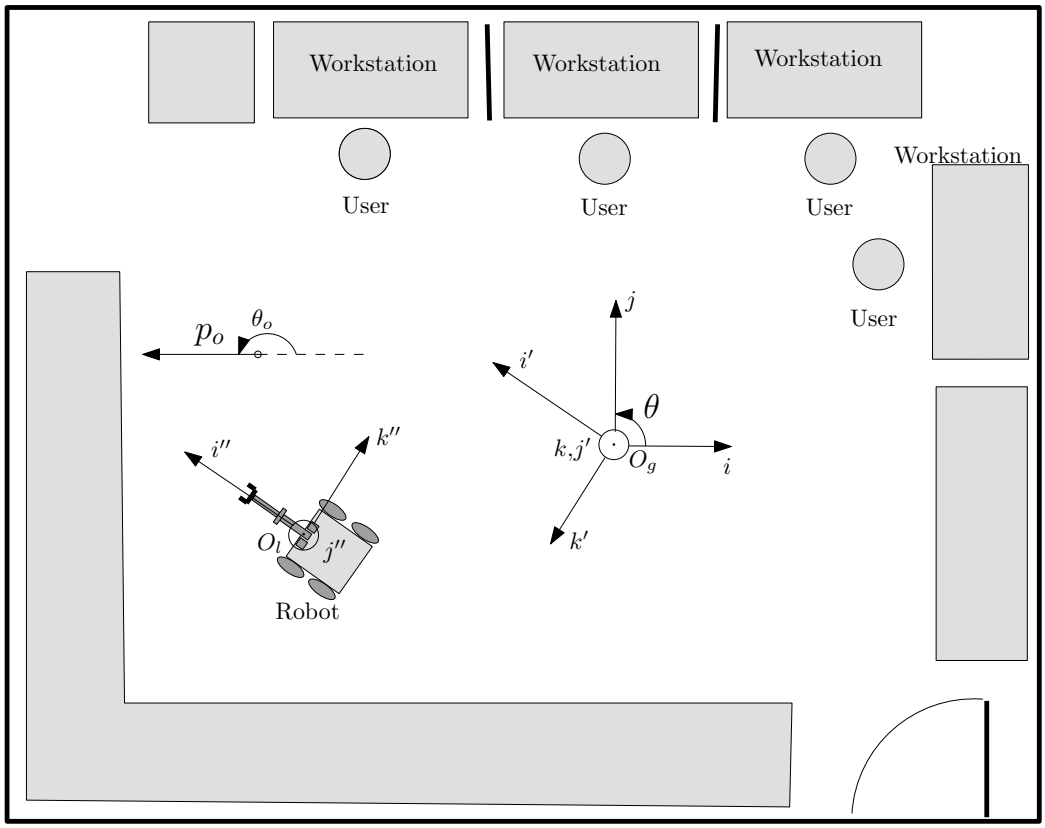


Figure 5.1: Coordinate frames: CF_{global} with unit vectors $(\mathbf{i}, \mathbf{j}, \mathbf{k})$ and origin O; CF_{local} with origin at O' and unit vectors $(\mathbf{i}'', \mathbf{j}'', \mathbf{k}'')$; and $CF_{rotated}$ with $(O, \mathbf{i}', \mathbf{j}', \mathbf{k}')$ as the origin and unit vectors respectively.

We use a third coordinate frame $CF_{rotated}$, with the origin O_g, that coincides

with the origin of the coordinate frame CF_{global}. The coordinate frame $CF_{rotated}$ is used to aid in the inverse dynamics calculations while calculating the robot's pose for a desired end-effector position in section 5.2.1. The unit vectors of $CF_{rotated}$ are parallel to those of CF_{global} with $\mathbf{k}' = \mathbf{j}'' = \mathbf{k}$, $\mathbf{i}' = \mathbf{i}''$ (figure 5.1).

5.1.2 The Robot

The physical robotic system that we use in our experiments is the "Corobot" developed by Coroware Inc. It is a mobile platform with an on-board robotic arm capable of reaching out and grasping objects (figure 5.2). The Corobot uses a skid-steer mechanism to change direction of motion. Three degrees of freedom are available in the arm, which moves in a plane perpendicular to the robots base: two to control the lower and upper limbs of the manipulator, and one degree of freedom provided at the wrist for a roll action. We assume the robot's base to be parallel to the ground at all times.

The corobot uses pre-written C and C++ libraries associated with the *Player project*. The Player environment is a user-interface that interacts with the sensors and motor encoders to drive the robot. To control the robot's base position, Player accepts linear and angular velocity commands (v, ω) and the kinematics of the base are $\dot{x} = v\cos\theta$, $\dot{y} = v\sin\theta$, $\dot{\theta} = \omega$. The arm can be controlled by specifying the coordinates of the end effector as (x', y', ϕ) where (x', y') are coordinates of the end-effector in the body-fixed coordinates CF_{local}, and ϕ is the orientation of the gripper. We fix the input ϕ to a constant value of 90^o or $\pi/2^c$ for picking up and delivering an object. Hence, ϕ remains constant and we will not use it during subsequent controller design.

5.1.3 Vicon^{TM} Motion Capture System

The position of the robot and the elements in the environment (the printer, print-out, etc.) are tracked using an infra-red motion capture system built by

Vicon^{TM}. This system consists of infrared cameras that capture real-time data from reflective markers glued to the surface of the robot. Different objects are distinguished from each other using different patterns of these infra-red markers on each object. The Vicon^{TM} system then returns the three dimensional position coordinates of each of the markers in a real-time stream. The coordinates are specified with respect to a user-specified global coordinate frame that can be calibrated and changed in any area within theirline-of-sight of the infrared cameras. Relevant position data from the Vicon^{TM} system is used by the corobot while running each controller, making it a closed-loop dynamical system.

5.2 Modeling a Mobile Manipulator

We denote q_p as the set of planar cartesian coordinates of the point on robot platform (base) where it is attached to the onboard arm. Hence $q_p \triangleq (x, y)$, and the angle of orientation of the base with respect to the global x-direction is denoted θ. This is the angle between the $\mathbf{i}$ and $\mathbf{i}'$ unit vectors as shown in figure 5.1.

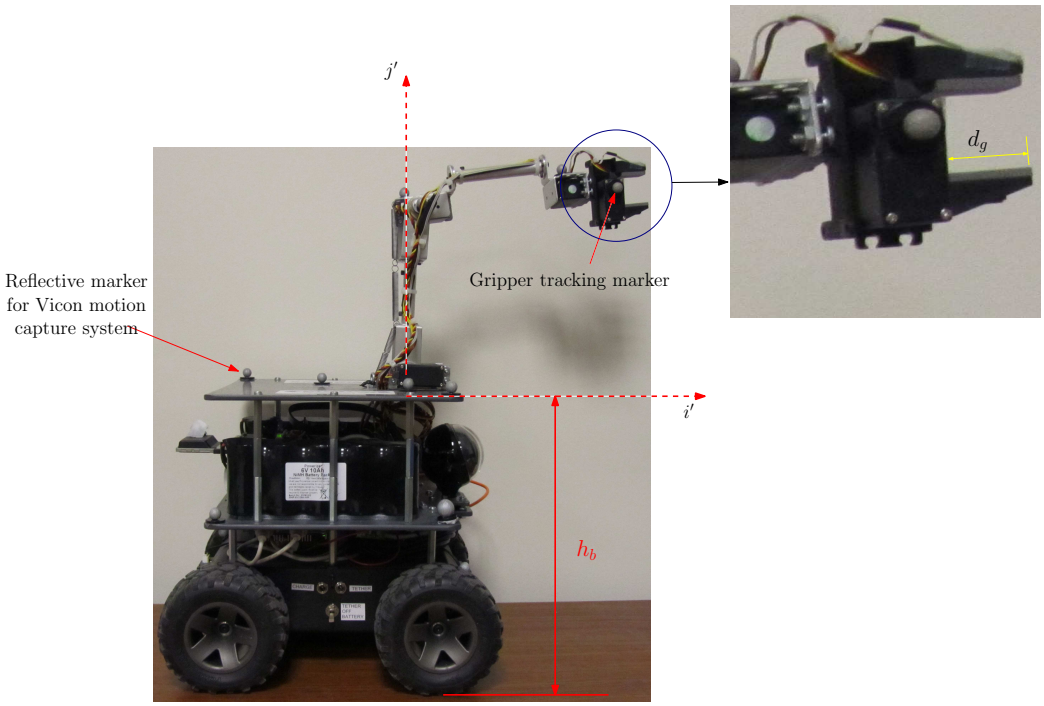


Figure 5.2: The Coroware Corobot used in our case study

The lengths of the lower and upper arm of corobot are denoted l_1 and l_2 respectively, and their angles with respect to the horizontal plane are denoted θ_1 and θ_2 respectively (see figure 5.3). The robot accepts two dimensional coordinate data x''_e, y''_e expressed in CF_{local}, as an input to steer its arm to the desired location. (The arm moves only along a plane with the z''_e coordinate remaining equal to zero.) The coordinates of the end effector with respect to CF_{local} are hence, $q_m \triangleq (x''_e, y''_e, 0)$. The points q_m lie in the workspace of the robot's arm denoted by $\mathcal{W}(q_p)$. The workspace of the arm is a function of the robot's base position since the workspace changes with respect to CF_{global} as the robot moves around.

To realize the motion and manipulation plan described in the example of section 3.4, we design three controllers; one for navigating the base, denoted as the textttgoto controller; and the other two `pick` and `place` for controlling the robot's arm to pick and place the stack of print-out.

- `goto`: The navigation controller u_a is designed by following a potential field approach. The input to the controller is the set of position and orientation values $p_p = (x_p, y_p, \theta_p)$ that form a parameter of the hybrid robotic system (see section 3.4 for the description of the hybrid system). The controller steers the robot to the specified point with the specified orientation upto within an allowable error captured as $\mathcal{B}_\varepsilon$, the ball of radius ε.

- `pick`: The arm controller u_b for picking up objects accepts the input $p_o = (x''_{oi}, y''_{oi}, z''_{oi}, x''_{of}, y''_{of}, z''_{oi})$ which specifies two (way)points defining a vector of arm motion to pick up an object, and forms another parameter for the hybrid system as explained in section 3.4. For the arm to be able to pick an object, the fore-arm of the robot should be aligned along the vector of approach from $(x''_{oi}, y''_{oi}, z''_{oi})$ to $(x''_{of}, y''_{of}, z''_{of})$, as specified by p_o. The controller grips the object upon reaching $(x''_{of}, y''_{of}, z''_{of})$ and returns to its pre-specified "home" position ${q^h_m}''$.

- `place`: The arm controller u_c for placing objects is very similar to the `pick` controller and uses the same parameter $p_o = (x''_{oi}, y''_{oi}, z''_{oi}, x''_{of}, y''_{of}, z''_{of})$ defining the final position $(x''_{of}, y''_{of}, z''_{of})$ to drop the object, as well as the initial position $(x''_{oi}, y''_{oi}, z''_{oi})$ to specify a vector of approach before the object is released. The difference between u_b and u_c is that u_c opens the gripper after reaching $(x''_{of}, y''_{of}, z''_{of})$, to release an object held in the gripper, while u_b closes the gripper to grasp an object. The gripper returns to its home position ${q^h_m}''$ after releasing the object.

The system described above can be brought into the form of the hybrid robotic agent of definition 3.1.1. For the Corobot, this system is $\mathbf{H}_r$ as defined in section 3.4. Section 3.4 also shows how a meaningful execution in $\mathbf{H}_r$ that completes the desired task, has a parallel run on the abstract finite transition system $\mathbf{T}_r$. We

show here how the process described in the example of section 3.4 is physically realized on the Corobot.

5.2.1 Inverse Kinematics

Before we proceed with showing how the plan sketched in section 3.4 can be executed on the Corobot to fetch a print-out, we have to introduce some mathematical preliminaries to find out the position and orientation q_p of the robot's base, so that it can grasp and release objects placed at a specified location p_o. The six-tuple p_o defines two points that determine the direction of gripper motion to pick or place objects in the workspace. These points, through inverse kinematics, can be used to calculate the desired position for the robot base q_p, which is subsequently used to assign a valuation to the predicate $\alpha_3 \Leftrightarrow p_o \in \mathcal{W}(q_p)$. This predicate is required to be true in order for the robot to be able to run the `pick` and `place` controllers as seen from table 3.1.

The inverse kinematics problem that we address here can be stated as follows. If $L \equiv (x_i, y_i, z_i)$ and $M \equiv (x_f, y_f, z_f)$ are two points defining a vector p_o in the global coordinate system of the robot's workspace, find the position and orientation of the robot base so that:

1. The two points L and M are in the workspace of the arm; i.e., $p_o \in \mathcal{W}(q_p)$.

2. The gripper remains parallel to the vector p_o as it moves along it.

The first condition requires that the robot be at an appropriate position and orientation q_p, such that the points p_o can be reached by the the robot's gripper. Since the arm moves in a plane, for the two points to be in $\mathcal{W}(q_p)$, the base should be aligned such that its arm moves in the same plane on which the points lie. Thus the base orientation θ must be θ_o as shown in figure 5.1.

For the second condition, the robot arm configuration must be similar to the configuration shown in figure 5.3. The figure shows the arm configuration in the

plane of the arm's motion, just before the `Pick` or `place` controllers can be activated. Essentially, the gripper must approach point (x_i, y_i, z_i) with the angle θ_3, as defined by the angle θ_2 that p_o makes with the horizontal. It must be noted that, with the base fixed, as the arm moves along vector p_o, the angle θ_3 will change. It must also be noted that the depth of the gripper d_g (see figure 5.2) provides an upper bound to the length of vector p_o to avoid interference of the object with the gripper. Then, since we have $p_o \leq d_g << l_1, l_2$, the variation in the angle θ_3 of arm's approach will remain small and can be ignored.

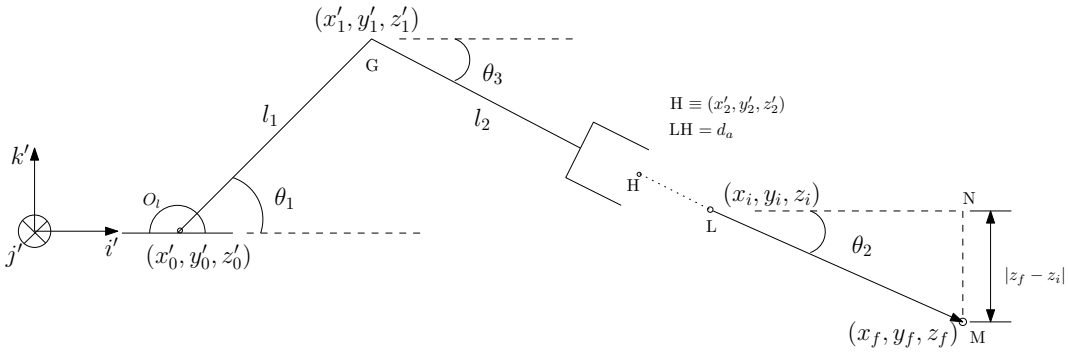


Figure 5.3: Figure showing alignment of Corobot's gripper with the approach vector p_o.

The inverse kinematics problem then reduces to finding three quantities:

1. The angle of orientation θ_o along which the robot must align itself, so that upon alignment $\theta = \theta_o$ (figure 5.1).

2. The angle of orientation (θ_2) of p_o in the vertical plane (figure 5.3). Upon alignment with the gripper, we would need $\theta_3 = \theta_2$ (the gripper is aligned with vector p_o) (figure 5.3).

3. The coordinates of the point x_o, y_o, z_o at which the origin O' of the robot must lie such that predicate α_3 is satisfied.

The desired angle of orientation for the robot is easy to calculate. On a plane perpendicular to the direction k, θ_o can be calculated as (see figure 5.1)

$$\theta_o = \tan^{-1}\left(\frac{y_f - y_i}{x_f - x_i}\right). \tag{5.1}$$

To find θ_2, considering triangle LMN in figure 5.3, we see that

$$\begin{aligned} &\sin\theta_2 = \tfrac{MN}{LM} \\ &\Leftrightarrow \theta_2 = \sin^{-1}\tfrac{|z_f - z_i|}{\sqrt{(y_f - y_i)^2 + (x_f - x_i)^2}}. \end{aligned} \tag{5.2}$$

Finally, we need the position O_l of the origin of the base on the robot. We start with finding the coordinates (x_2', y_2', z_2') of point H, assumed to be the end point of the gripper (figure 5.3). We assume that point H is desired at a distance of d_a from the point L (the starting piont of the vector p_o) (figure 5.3). To proceed with our calculations, it is convinient to use the coordinate frame $CF_{rotated}$. As a reminder, quantities expressed with a prime notation, e.g., (x_2', y_2', z_2'), represent the values of H expressed in CF_{local}, whereas the same quantities without a prime notation, i.e., (x_2, y_2, z_2), are the corresponding values of H in global coordinates CF_{global}.

Knowing the values of θ_o and θ_2 from equations (5.2) and (5.1), we proceed as follows. The coordinates of point L in $CF_{rotated}$ are computed as

$$\begin{bmatrix} x_i' \\ y_i' \\ z_i' \end{bmatrix} = R \cdot \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}. \tag{5.3}$$

where R is the rotation matrix

$$R = \begin{bmatrix} i.i' & j.i' & k.i' \\ i.j' & j.j' & k.j' \\ i.k' & j.k' & k.k' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ 0 & 0 & 1 \\ -\sin\theta & \cos\theta & 0 \end{bmatrix}. \tag{5.4}$$

Next, for $\theta = \theta_o$ and $\theta_3 = \theta_2$, the coordinates of point H are computed geometrically as (figure 5.3)

$$\begin{aligned} y'_2 &= y'_i \\ z'_2 &= z'_i + d_a \cdot \sin\theta_2 \\ x'_2 &= x'_i - d_a \cdot \cos\theta_2. \end{aligned} \tag{5.5}$$

Denoting h_b the height of base from the ground (figure 5.2), the coordinates of O_l can be found as:

$$\begin{aligned} y'_0 &= y'_1 = y'_2 \\ z'_0 &= h_b \\ z'_1 &= z'_2 + l_2 \cdot \sin\theta_3 \\ x'_1 &= x'_2 - l_2 \cdot \cos\theta_3 \\ \theta_1 &= \sin^{-1}\tfrac{(z'_1 - z'_0)}{l_1} \\ x'_0 &= x'_1 - l_1 \cdot \cos\theta_1. \end{aligned} \tag{5.6}$$

These coordinates of O_l can be found in CF_{global} by using the coordinate transformation:

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = R^T \cdot \begin{bmatrix} x'_0 \\ y'_0 \\ z'_0 \end{bmatrix}. \tag{5.7}$$

5.2.2 Simple Tasks: Fetching a Print-out

With our experiments, we want to show how an appropriate switching stategy for in a transition system can be implemented on a underlying hybrid system such as the Corobot robotic platform. To show this, we encode each controller, u_a, u_b

and u_c as C++ functions that accept the parameters p_p for controller u_a, and p_m for the arm controllers u_b and u_c as input. These C++ functions implement the part of the hyrbid system $\mathbf{H}_r$ that closes the loop around the continuous dynamics. In the main body of the algorithm, the function calls give rise to controller sequencing and transitions in the transition system $\mathbf{T}_r$.

Since transitions in system $\mathbf{T}_r$ have a parallel run in $\mathbf{H}_r$, we plan our task using the transition system $\mathbf{T}_r$. To fetch a printout, the pseudo-code of the main program looks like the following:

START	% Marks the beginning of the main program.
$u_a(p_p^r)$	% $p_p^r = (x_p^r, y_p^r, \theta_p^r)$ is the robot's base position at the printer.
$u_b(p_o^r)$	% $p_o^r = L^r M^r$ is the vector along the stack of print-out.
$u_a(p_p^u)$	% $p_p^r = (x_p^u, y_p^u, \theta_p^u)$ is the base position at user location.
$u_c(p_o^r)$	% $p_o^u = L^u M^u$ is the vector defining the delivery place for the print-out.
$u_a(p_p^h)$	% $p_p^h = (x_p^r, y_p^r, \theta_p^r)$ is the parking position for the corobot.
STOP	% End of program

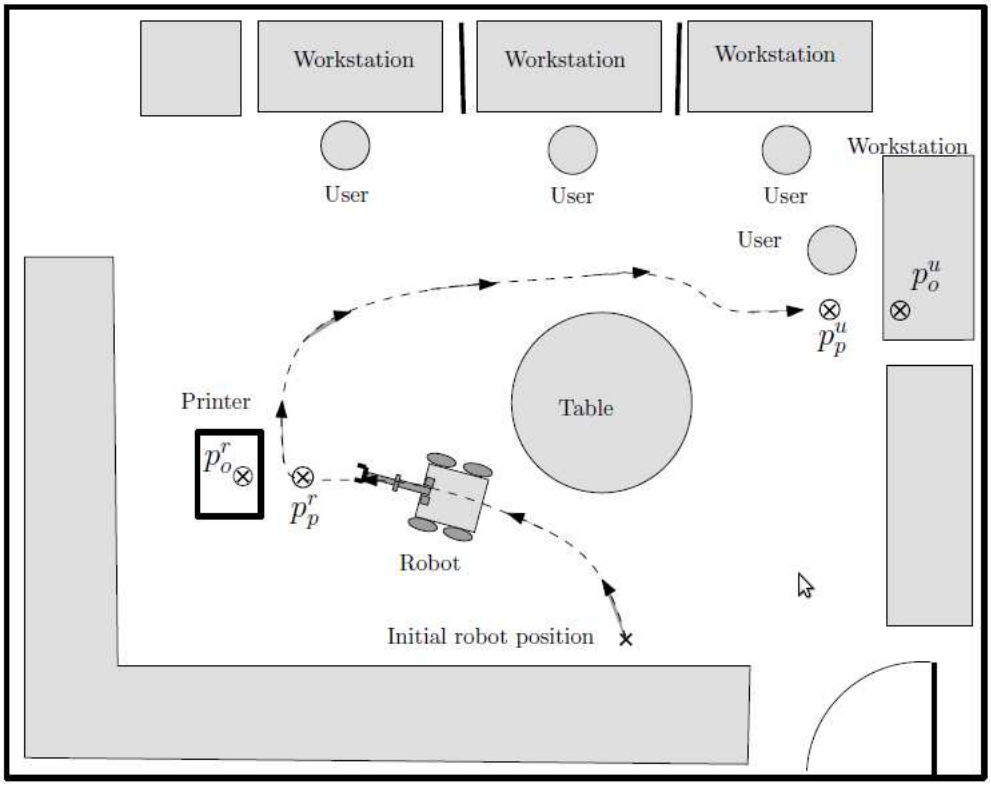


Figure 5.4: The "fetch the printout" scenario, implemented on a Corobot.

The experiments were conducted in the Cooperative Robotics Lab of the University of Delaware, the floor plan of which looks like the one in figure 5.4 [1]. To fetch a print-out from the printer located at $p_p^r = (x_p^r, y_p^r, \theta_p^r)$, the controller u_a is run by a function call that executes the closed loop dynamics for navigation. After the robot reaches the printer location, the post-condition of the `goto` controller is satisfied and the C++ function for u_a returns the command to the main program. Next, the parameter is reset to p_o^r so that the pre-condition of controller u_b is satisfied, and the robot picks up the stack of print-out, thereafter returning the command from the u_b function to the main program. The program proceeds with the sequence $u_a\, u_c\, u_a$ to deliver the print-out and return to its parking location.

[1] This is the same as figure 3.1

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

In this work we start with a hybrid dynamical system with asymptotically stable dynamics and parameterized attractors that can be abstracted to a purely discrete, finite transition system. We show that the discrete transition system is weakly bisimilar to the hybrid system it originated from. This result allows motion planning and behavior design for the hybrid system to be performed on the discrete system, without concerns about the continuous dynamics of the former.

Having the continuous dynamics of the hybrid system abstracted, we can focus our attention to sequences of controllers that lead to an overall behavior of the system. The finite transition system, induced by the hybrid system, can be represented by a directed graph such as a finite state automaton that accepts a regular language of sequences of controllers that can be executed on the hybrid system. Controller sequencing is possible only when the Pre conditions necessary for the execution of each controller are satisfied by the Post conditions of the preceeding controller(s) in the sequence. Local and long-distance constraints arise due to such Pre and Post condition compatibility between controllers, and each constraint defines a class of language that can be analyzed using formal language theory. We identify that the local constraints for controller concatenation can be captured by strictly local languages of the Sub-regular hierarchy, while the long-distance constraints are captured by, what we define as, the tier-based strictly local languages.

An intersection of languages from these two classes give us regular languages of strings of controller sequences that can be executed on the hybrid system. We call this class of languages, *regular robotic languages*. Regular languages exhibit superior decidability and closure results than other language classes in the Chomsky hierarchy, which offer computational advantages when performing operations with them.

In addition to showing that the classes of robotic languages are (sub)regular, this work identifies that regular robotic languages belong to the star free class of the Sub-regular hierarchy. Such a result identifies a unique domain of existing results on closure properties and set operations corresponding to star free languages, that can be applied to regular robotic languages as well. Such findings open doors for future work on studying compositions, decompositions and language learning using regular robotic languages.

6.2 Future Work

Several subregular languages including the strictly local languages have been studied in terms of admitting the construction of efficient algorithms (learners) that can identify them in the limit from positive data [25]. These learning algorithms are promising for identifying languages in the class that we study, especially if we apply them to the local and tier-based local language patterns. Showing that regular robotic languages are learnable in a string extension learning framework offers an immediate future area of research with the benefits of enabling a robotic system to construct models of external processes in its environment by observing a sequence of events in it. This work would be a step toward further automation of a planning and control design synthesis.

Furthermore, the concepts of languages and automata, presented in this work, pave the way for an algebraic analysis of regular robotic languages. A mathematical study of the finite state machines for the subregular patterns would give new insights

on the kinds of mathematical and set operations that can be performed on such patterns. Such mathematical analysis of hybrid systems from a viewpoint of the controller strings they admit, can be useful for composing and decomposing models that describe behaviors of groups of homogeneous or heterogeneous robotic systems, working in teams.

Finally, we were able to identify some parallels between regular robotic languages and autosegmental patterns seen in computational phonology. A deeper study of autosegmental phonology can help apply trajectory optimization rules on strings of hybrid system controllers, based on sound-pattern optimizations studied by linguists.

BIBLIOGRAPHY

[1] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[2] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, July 2000.

[3] Rajeev Alur, Thao Dang, and Franjo Ivančić. Predicate abstraction for reachability analysis of hybrid systems. *ACM Transactions on Embedded Computer Systems*, 5:152–199, February 2006.

[4] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, E. Klavins, and G.J. Pappas. Symbolic planning and control of robot motion. *Robotics and Automation Magazine, IEEE*, 14(1):61–70, 2007.

[5] C. Belta, V. Isler, and G.J. Pappas. Discrete abstractions for robot motion planning and control in polygonal environments. *IEEE Transactions on Robotics*, 21(5):864–874, 2005.

[6] R.W. Brockett. On the computer control of movement. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 534–540 vol.1, April 1988.

[7] W.L. Brogan. *Modern Control Theory*. Prentice Hall, 2 edition, 1991.

[8] Christos G. Cassandras and Stephane Lafortune. *Introduction to Discrete Event Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[9] N. Chomsky. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3):113–124, September 1956.

[10] Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

[11] D.C. Conner, A.A. Rizzi, and H. Choset. Composition of local potential functions for global robot control and navigation. In *Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 4, pages 3546–3551 vol.3, 2003.

[12] R.A. Decarlo, M.S. Branicky, S. Pettersson, and B. Lennartson. Perspectives and results on the stability and stabilizability of hybrid systems. *Proceedings of the IEEE*, 88(7):1069–1082, July 2000.

[13] Magnus Egerstedt. *Motion Description Languages for Multi-Modal Control in Robotics*, volume 4, pages 75–89. Springer Berlin / Heidelberg, 2003.

[14] G.E. Fainekos, H. Kress-Gazit, and G.J. Pappas. Temporal logic motion planning for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2020–2025, 2005.

[15] E. Frazzoli and F. Bullo. On quantization and optimal control of dynamical systems with symmetries. In *Proceedings of the IEEE Conference on Decision and Control*, volume 1, pages 817–823, Las Vegas, NV, December 2002.

[16] E. Frazzoli, M. A. Dahleh, and E. Feron. Maneuver-based motion planning for nonlinear systems with symmetries. *IEEE Transactions on Robotics*, 21(6):1077–1091, December 2005.

[17] Paul Gastin and Denis Oddoux. Fast LTL to büchi automata translation. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer Berlin / Heidelberg, 2001.

[18] A. Girard and G.J. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.

[19] A. Girard and G.J. Pappas. Hierarchical control system design using approximate simulation. *Automatica*, 45(2):566–571, 2009.

[20] Antoine Girard and Colas Le Guernic. Zonotope/hyperplane intersection for hybrid systems reachability analysis. In Magnus Egerstedt and Bud Mishra, editors, *Hybrid Systems: Computation and Control*, volume 4981 of *Lecture Notes in Computer Science*, pages 215–228. Springer Berlin / Heidelberg, 2008.

[21] John Anton Goldsmith. *Autosegmental phonology*. Ph.D. dissertation, Massachusetts Institute of Technology. Department of Foreign Literatures and Linguistics., 1976.

[22] Susanne Graf and Hassen Saidi. Construction of abstract state graphs with PVS. In Orna Grumberg, editor, *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer Berlin / Heidelberg, 1997.

[23] Sumit Gulwani and Ashish Tiwari. Constraint-based approach for analysis of hybrid systems. In Aarti Gupta and Sharad Malik, editors, *Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 190–203. Springer Berlin / Heidelberg, 2008.

[24] M.A. Harrison. *Introduction to formal language theory*, volume 312. Addison-Wesley, 1978.

[25] Jeffrey Heinz. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 897–906, Uppsala, Sweden, July 2010. Association for Computational Linguistics.

[26] Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. Tier-based strictly local constraints for phonology. Submitted, 2011.

[27] J.P. Hespanha. Uniform stability of switched linear systems: extensions of LaSalle's invariance principle. *IEEE Transactions on Automatic Control*, 49(4):470–482, 2004.

[28] Gerard J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23:279–295, May 1997.

[29] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison Wesley, 2 edition, November 2000.

[30] John E. Hopcroft and Jeffrey D. Ullman. *Formal languages and their relation to automata.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1969.

[31] D. Hristu-Varsakelis, M. Egerstedt, and P.S. Krishnaprasad. On the structural complexity of the motion description language MDLe. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, volume 4, pages 3360–3365 vol.4, 2003.

[32] H.K. Khalil. *Nonlinear systems*, volume 3. Prentice Hall New Jersey, 2002.

[33] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[34] Marius Kloetzer and Calin Belta. A fully automated framework for control of linear systems from ltl specifications. In João Hespanha and Ashish Tiwari, editors, *Hybrid Systems: Computation and Control*, volume 3927 of *Lecture Notes in Computer Science*, pages 333–347. Springer Berlin / Heidelberg, 2006.

[35] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1398–1404 vol.2, April 1991.

[36] B.H. Krogh. A generalized potential field approach to obstacle avoidance control. In *Proceedings of the International Robotics Control Conference*, pages 1150–1156, 1984.

[37] G. Lafferriere, G. J. Pappas, and S. Yovine. *A new class of decidable hybrid systems*, volume 1569. Springer, 1999.

[38] D. Liberzon. *Switching in systems and control*. Springer, 2003.

[39] J. Lygeros. Lecture notes on hybrid systems. In *Notes for an ENSIETA workshop*, 2004.

[40] V. Manikonda, P.S. Krishnaprasad, and J. Hendler. A motion description language and a hybrid architecture for motion planning with nonholonomic robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 2021–2028, May 1995.

[41] Kenneth L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Ph.D. dissertation, Carnegie Mellon University, Department of Computer Science, May 1992.

[42] Robert McNaughton and Seymour A. Papert. *Counter-Free Automata*. Series#65. The MIT Press, 1971.

[43] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[44] Robin Milner. An algebraic definition of simulation between programs. In *Proceedings of the 2nd International Conference on Artificial Intelligence*, pages 481–489, Stanford, CA, USA, September 1971. Stanford University.

[45] Robin Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1st edition, June 1999.

[46] K. Ogata. *Modern control engineering*. Prentice Hall, 2009.

[47] G.J. Pappas. Bisimilar linear systems. *Automatica*, 39(12):2035–2047, 2003.

[48] S. Pettersson and B. Lennartson. Stability and robustness for hybrid systems. In *Proceedings of the 35th IEEE Conference on Decision and Control*, volume 2, pages 1202–1207 vol.2, December 1996.

[49] S. Pettersson and B. Lennartson. Stabilization of hybrid systems using a min-projection strategy. In *Proceedings of the 2001 American Control Conference*, 2001.

[50] J.L. Piovesan, H.G. Tanner, and C.T. Abdallah. Discrete asymptotic abstractions of hybrid systems. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 917–922, 2006.

[51] André Platzer and Edmund Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Formal Methods in System Design*, 35:98–120, 2009.

[52] Chetan Rawal, Herbert G. Tanner, and Jeffrey Heinz. (Sub)regular robotic languages. Submitted, 2011.

[53] E. Rimon and D.E. Koditschek. Exact robot navigation using artificial potential functions. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, October 1992.

[54] J. Rogers and G.K. Pullum. Aural pattern recognition experiments and the subregular hierarchy. In *Proceedings of 10th Mathematics of Language Conference*, pages 1–7, 2007.

[55] James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language*, volume 6149 of *Lecture Notes in Computer Science*, pages 255–265. Springer Berlin / Heidelberg, 2010.

[56] Imre Simon. Piecewise testable events. In H. Brakhage, editor, *Automata Theory and Formal Languages*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer Berlin / Heidelberg, 1975.

[57] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1996.

[58] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In E. Emerson and A. Sistla, editors, *Computer Aided Verification*, volume 1855 of *Lecture Notes in Computer Science*, pages 248–263. Springer Berlin / Heidelberg, 2000.

[59] Olaf Stursberg and Bruce H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In O. Maler and A. Pnueli, editors, *Proceedings of the 6th international conference on Hybrid systems: computation and control*, pages 482–497, Berlin, Heidelberg, 2003. Springer-Verlag.

[60] P. Tabuada. Symbolic sub-systems and symbolic control of linear systems. In *44th IEEE Conference on Decision and Control.*, pages 18–23, 2005.

[61] P. Tabuada. Approximate simulation relations and finite abstractions of quantized control systems. *Hybrid Systems: Computation and Control*, pages 529–542, 2007.

[62] P. Tabuada and G.J. Pappas. Linear time logic control of discrete-time linear systems. *IEEE Transactions on Automatic Control*, 51(12):1862–1877, 2006.

[63] H. Tanner and G.J. Pappas. Simulation relations for discrete-time linear systems. In *Proceedings of the IFAC World Congress on Automatic Control*, pages 1302–1307, 2002.

[64] Herbert G. Tanner, Chetan Rawal, Jie Fu, Jorge L. Piovesan, and Chaouki T. Abdallah. Finite asymptotic abstractions for hybrid systems with stable continuous dynamics. Discrete Event Dynamic Systems (submitted) 2010.

[65] Y. Tazaki and J. Imura. Finite Abstractions of Discrete-time Linear Systems and Its Application to Optimal Control. In *17th IFAC World Congress*, pages 10201–10206, 2008.

[66] C.J. Tomlin, I. Mitchell, A.M. Bayen, and M. Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003.

[67] Moshe Vardi. An automata-theoretic approach to linear temporal logic. In Faron Moller and Graham Birtwistle, editors, *Logics for Concurrency*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer Berlin / Heidelberg, 1996.

[68] Moshe Y. Vardi. *An automata theoretic approach to automatic program verification.* Yorktown Heights, N.Y. : International Business Machines Inc., Thomas J. Watson Research Center, 1986.

[69] Sheng Yu, Qingyu Zhuang, and Kai Salomaa. The state complexities of some basic operations on regular languages. *Theoretical Computer Science*, 125(2):315–328, 1994.